Effective Policies for Workflow Scheduling in Stochastic Environments

Michał Wośko

Submitted in fulfilment of the requirements for the degree of Doctor of Philosophy

Faculty of Science, Engineering and Technology
Swinburne University of Technology

2015
to my mom and friends
Abstract

There exist many realistic workflow scheduling optimization problems that have been insufficiently tackled by research or have been solved under restrictive and simplified assumptions. Most scheduling policies support only directed acyclic graphs, or even more restricted application models, with deterministic activity times. Some only allow stochastic task execution times but not inter-task communication delays, which are associated with data transfers between tasks executing on different processors. When stochastic activities are considered, most approaches make additional unrealistic assumptions and only admit specific probability distributions. Very few consider problems with multiple workflow instances, especially when this multiplicity results from a stochastic arrival process. Others support multiple job sets with no or only rudimentary structure. On the other hand, many realistic workflows feature conditional execution, corresponding to choices, and iteration of activities. Such workflow models are investigated by very few scheduling research efforts.

This thesis fills the gaps outlined above. It provides solutions to a range of problems related to scheduling the atomic activities of workflows onto a set of heterogeneous processors and communication channels. These solutions rely on a set of novel, accurate methods of estimating the values of the stochastic variables of interest, in a manner that takes into account the requirements of an efficient scheduling process. Activities with completely arbitrary probability distributions are supported.

In successive chapters, increasingly complex problem settings are considered. At first, for a single instance of a directed acyclic task graph with stochastic activities, an offline scheduling policy is formulated. It is based on a novel approach, parameter-driven greediness, which schedules jobs of lower priority when this is unlikely to delay higher priority ones or the projected gain is significant in comparison. Within the policy, the values of the stochastic variables are estimated with a novel method, based on the partial moments of these variables. Next, a setting is examined whose distinctive feature is a queue of workflow instances, resulting from an arbitrary stochastic arrival process. The solution is an online policy which balances the conflicting objectives of minimizing the schedule length of the instances currently under processing and of the waiting time of new ones. New instances are started when this decision is estimated to cause a negligible delay to the completion of those under processing. Finally, the workflow net
application model is considered, which allows for the specification of workflows with conditional execution and iteration. An efficient algorithm is put forward to determine the set of schedulable jobs at each decision moment. Efficient and accurate methods are proposed to prioritize jobs that depend on arbitrary combinations of paths containing stochastic activities. These methods support conditional branches, corresponding to both exclusive and non-exclusive choices, and combinations of loops, with finite or potentially infinite iterations. The accuracy of these estimates is additionally improved for paths containing activities which are running at the moment the scheduler performs the estimation.

All the policies proposed are thoroughly evaluated experimentally with respect to their performance and robustness. All methods applied to the estimation of stochastic activity times and of path lengths are also tested individually.
I would like to thank the Smart Services CRC for funding this work. Throughout my candidature, the Smart Services CRC has provided me also with invaluable advice, logistic and organizational support in difficult moments, but also with training and opportunities of fruitful exchange with other researchers and industry partners.

I would like to thank my Supervisors, Dr. Irene Moser, Dr. James Montgomery and Dr. hab. Andrzej Zbrzezny. My deepest gratitude goes to Irene, who has helped me through her feedback, but also motivated and supported me, often going out of her way in doing so.

I wish to thank my colleagues, especially at Swinburne and Wrocław University of Technology and Jan Długosz University. They have been with me in this journey, in its marvellous moments and in the less beautiful ones. They have supported me both with research-related input and feedback and on a personal level. I thank Dr. Marek Woda, whom I am proud to consider my friend and who made it possible for me to start this journey. I thank Dr. Alireza Tashakori Abkenar, who has kept me continuing on this path, has always supported me in the most difficult moments and has become my dearest friend. I thank my great friend Dr. Khalid Mansour, who has worked with me on many research questions, but more than that has been a source of inspiration and optimism. I am especially grateful to Ms. Elena Dubinina for her extraordinary support and for always being there to help. I thank Ms. Idalina Alcantara for always being courageous and giving me courage. I thank Dr. Grzegorz Popek, Dr. Aldeida Aleti-Watne, Dr. Chiara Paviolo, Ms. Gillian Foster, Ms. Agnieszka Zbrzezny, Ms. Dominika Kubicka, Ms. Anna Czekalska, Dr. Syed Ghalib, Dr. Jan Richter, Dr. Ingo Müller and Ms. Li Sim Loh for all their invaluable help and for being my friends. I am grateful to all my friends and family members. All of them have inspired me in many ways and have given me confidence. For the inspiration I want to especially thank my late uncle, Mr. Zenon Horodyski, whose talent and brave spirit will always be alive.

Last but certainly not least, I want to express my gratitude to my mother, Teresa Wośko, a strong but incredibly gentle person. Without her support and wisdom this achievement would have not been possible. I am happy to share it with her.
Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma, except where due reference is made. To the best of my knowledge, this thesis contains no material previously published or written by another person except where due reference is made in the text of the thesis.

________________________________________

Michał Wośko

________________________________________

30/01/2015
Contents

Abstract ii

Acknowledgements iv

Declaration v

1 Introduction 1
  1.1 Research Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
  1.2 Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . 6
  1.3 Thesis Overview . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

Publications 11

2 Background and Preliminaries 12
  2.1 Problem Context . . . . . . . . . . . . . . . . . . . . . . . . . . . . 12
  2.2 Scheduling Framework . . . . . . . . . . . . . . . . . . . . . . . . . 13
    2.2.1 Basic Application Model . . . . . . . . . . . . . . . . . . . . 14
    2.2.2 Scheduling Platform . . . . . . . . . . . . . . . . . . . . . . 16
  2.3 Schedule . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 18
    2.3.1 Equivalences . . . . . . . . . . . . . . . . . . . . . . . . . . . 20
    2.3.2 Start and Finish Time of Jobs . . . . . . . . . . . . . . . . . 21
  2.4 Scheduling Policies . . . . . . . . . . . . . . . . . . . . . . . . . . . 24
  2.5 Schedule Performance and Optimality . . . . . . . . . . . . . . . . . 26
  2.6 Relevant Functions in Single-Objective Scheduling . . . . . . . . . . 27
  2.7 Multi-Objective Scheduling . . . . . . . . . . . . . . . . . . . . . . 28
  2.8 Notable Issues in Bi-Objective Scheduling . . . . . . . . . . . . . . . 30
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.9</td>
<td>Probability-Theoretical Background</td>
<td>33</td>
</tr>
<tr>
<td>2.9.1</td>
<td>Relevant Properties of Independent Random Variables</td>
<td>36</td>
</tr>
<tr>
<td>2.10</td>
<td>Performance Space and Optimality in Scheduling</td>
<td>37</td>
</tr>
<tr>
<td>2.11</td>
<td>Scheduling as a Sequential Decision Process</td>
<td>39</td>
</tr>
<tr>
<td>2.12</td>
<td>Taxonomy in Scheduling for Parallel Processing</td>
<td>43</td>
</tr>
<tr>
<td>2.13</td>
<td>Offline and Online Scheduling Models</td>
<td>46</td>
</tr>
<tr>
<td>2.14</td>
<td>Optimization of Schedule Length</td>
<td>48</td>
</tr>
<tr>
<td>2.15</td>
<td>Evaluation of Schedule Length</td>
<td>50</td>
</tr>
<tr>
<td>2.15.1</td>
<td>Related Work in Program Evaluation and Review Technique</td>
<td>51</td>
</tr>
<tr>
<td>2.16</td>
<td>List Scheduling</td>
<td>52</td>
</tr>
<tr>
<td>2.16.1</td>
<td>Basic Techniques for Priority Determination</td>
<td>53</td>
</tr>
<tr>
<td>2.16.2</td>
<td>Notable Generic List Scheduling Algorithms</td>
<td>57</td>
</tr>
<tr>
<td>2.17</td>
<td>Uncertainty and Robustness in Scheduling</td>
<td>64</td>
</tr>
<tr>
<td>2.18</td>
<td>Queuing Theory</td>
<td>68</td>
</tr>
<tr>
<td>2.19</td>
<td>Experimental Evaluation</td>
<td>71</td>
</tr>
<tr>
<td>2.20</td>
<td>Summary</td>
<td>73</td>
</tr>
<tr>
<td>3</td>
<td>Offline Scheduling of DAG-reducible Stochastic Workflows</td>
<td>77</td>
</tr>
<tr>
<td>3.1</td>
<td>Motivating Scenarios</td>
<td>78</td>
</tr>
<tr>
<td>3.2</td>
<td>Problem Formulation</td>
<td>82</td>
</tr>
<tr>
<td>3.3</td>
<td>Related Work</td>
<td>83</td>
</tr>
<tr>
<td>3.4</td>
<td>Problems with Existing Solutions</td>
<td>86</td>
</tr>
<tr>
<td>3.5</td>
<td>Solution Approach</td>
<td>89</td>
</tr>
<tr>
<td>3.5.1</td>
<td>Partial Moments in Path Length Estimation</td>
<td>89</td>
</tr>
<tr>
<td>3.5.2</td>
<td>The Stochastic Trade-off Insertion Mechanism</td>
<td>92</td>
</tr>
<tr>
<td>3.5.3</td>
<td>Complexity Analysis</td>
<td>97</td>
</tr>
<tr>
<td>3.6</td>
<td>Experimental Evaluation</td>
<td>98</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Discussion</td>
<td>104</td>
</tr>
<tr>
<td>3.7</td>
<td>Summary</td>
<td>118</td>
</tr>
<tr>
<td>4</td>
<td>Online Scheduling of DAG-reducible Stochastic Workflows</td>
<td>120</td>
</tr>
<tr>
<td>4.1</td>
<td>Motivating Scenarios</td>
<td>122</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Motivation for Response Time Optimization</td>
<td>123</td>
</tr>
<tr>
<td>4.2</td>
<td>Problem Formulation</td>
<td>124</td>
</tr>
</tbody>
</table>
# List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Task graph of the anti-lock breaking system (ABS, following [7]).</td>
<td>79</td>
</tr>
<tr>
<td>3.2</td>
<td>Functional view of the cloud application scenario.</td>
<td>80</td>
</tr>
<tr>
<td>3.3</td>
<td>A possible workflow schema of the cloud application scenario from Figure 3.2.</td>
<td>80</td>
</tr>
<tr>
<td>3.4</td>
<td>A simplified skilled migrant visa processing workflow.</td>
<td>81</td>
</tr>
<tr>
<td>3.5</td>
<td>Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s critical path (CP), as a function: (a) of the processor multiplier $p_{mul}$, given $</td>
<td>P</td>
</tr>
<tr>
<td>3.6</td>
<td>(c) Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $nsd = \frac{\sigma}{\mu}$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $nsd$. Algorithms with variable greediness.</td>
<td>106</td>
</tr>
<tr>
<td>3.7</td>
<td>Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s critical path (CP), as a function: (a) of the processor multiplier $p_{mul}$, given $</td>
<td>P</td>
</tr>
<tr>
<td>3.8</td>
<td>(c) Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $nsd = \frac{\sigma}{\mu}$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $nsd$. The proposed trade-off-based algorithm with different stochastic estimators.</td>
<td>109</td>
</tr>
</tbody>
</table>
List of Figures

3.9 Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s critical path (CP), as a function: (a) of the processor multiplier $p_{mul}$, given $|P| = |vp_{mul}|$; (b) of the computation-to-communication ratio ($ccr$). Algorithms with different job prioritizing and machine assignment heuristics. ........................................ 110

3.10 (c) Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $nsd = \frac{\sigma}{\mu}$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $nsd$. Algorithms with different job prioritizing and machine assignment heuristics. ........................................ 111

3.11 Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s critical path (CP), as a function: (a) of the processor multiplier $p_{mul}$, given $|P| = |vp_{mul}|$; (b) of the computation-to-communication ratio ($ccr$). The proposed trade-off-based algorithm with different processor and channel speed normalization strategies. .................. 113

3.12 (c) Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $nsd = \frac{\sigma}{\mu}$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $nsd$. The proposed trade-off-based algorithm with different processor and channel speed normalization strategies. .................. 114

3.13 Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s critical path (CP), as a function: (a) of the processor multiplier $p_{mul}$, given $|P| = |vp_{mul}|$; (b) of the computation-to-communication ratio ($ccr$). Algorithms with different insertion strategies. .................. 116

3.14 (c) Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $nsd = \frac{\sigma}{\mu}$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $nsd$. Algorithms with different insertion strategies. .................. 117

4.1 A typical three-layered web application. ........................................ 123

4.2 Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s critical path (CP), as a function: (a) of the processor multiplier $p_{mul}$, given $|P| = |vp_{mul}|$; (b) of the multiplier $\lambda_{mul}$ of the critical arrival rate, given $\lambda = \lambda_{mul}\lambda_{max}$. Algorithms with variable greediness. .................. 147
List of Figures

4.3 (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation \( nsd = \frac{\sigma}{\mu} \) of the activity durations; (d) standard deviations of the normalized response time, as a function of \( nsd \). Algorithms with variable greediness. 148

4.4 Sample mean and dispersion of the response time for an instance, normalized by the processing time of all the task realizations in the instance’s critical path, as a function: (a) of the processor multiplier \( p_{mul} \), given \( |P| = \lfloor vp_{mul} \rfloor \); (b) of the multiplier \( \lambda_{mul} \) of the critical arrival rate, given \( \lambda = \lambda_{mul} \lambda_{max} \). The proposed trade-off-driven algorithm with OBN delay, working with different stochastic estimators. 150

4.5 (c) Sample mean and dispersion of the response time for an instance, normalized by the processing times of the task realizations in the instance’s CP, as a function of the normalized standard deviation \( nsd = \frac{\sigma}{\mu} \) of the activity durations; (d) standard deviations of the normalized response time, as a function of \( nsd \). The proposed trade-off-driven algorithm with OBN delay, working with different stochastic estimators. 151

4.6 Sample mean and dispersion of the response time for an instance, normalized by the processing time of all the task realizations in the instance’s critical path, as a function: (a) of the processor multiplier \( p_{mul} \), given \( |P| = \lfloor vp_{mul} \rfloor \); (b) of the multiplier \( \lambda_{mul} \) of the critical arrival rate, given \( \lambda = \lambda_{mul} \lambda_{max} \). Algorithms with different job prioritizing and machine assignment heuristics. 153

4.7 (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation \( nsd = \frac{\sigma}{\mu} \) of the activity durations; (d) standard deviations of the normalized response time, as a function of \( nsd \). Algorithms with different job prioritizing and machine assignment heuristics. 154

4.8 Sample mean and dispersion of the response time for an instance, normalized by the processing time of all the task realizations in the instance’s critical path, as a function: (a) of the processor multiplier \( p_{mul} \), given \( |P| = \lfloor vp_{mul} \rfloor \); (b) of the multiplier \( \lambda_{mul} \) of the critical arrival rate, given \( \lambda = \lambda_{mul} \lambda_{max} \). Algorithms with different processor and channel speed normalization strategies. 156

4.9 (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation \( nsd = \frac{\sigma}{\mu} \) of the activity durations; (d) standard deviations of the normalized response time, as a function of \( nsd \). Algorithms with different processor and channel speed normalization strategies. 157
List of Figures

4.10 Sample mean and dispersion of the response time for an instance, normalized by the processing time of all the task realizations in the instance’s critical path, as a function: (a) of the processor multiplier $p_{mul}$, given $|P| = \lceil vp_{mul} \rceil$; (b) of the multiplier $\lambda_{mul}$ of the critical arrival rate, given $\lambda = \lambda_{mul}\lambda_{max}$. Algorithms with different cross-instance strategies. ........................................ 158

4.11 (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $nsd = \frac{\sigma}{\mu}$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $nsd$. Algorithms with different cross-instance strategies. .......................... 159

5.1 A possible DIAC skilled migrant visa processing workflow with a conditional control flow [269]. The circles represent conditions evaluated during execution, based on the output of the preceding tasks. The evaluation output (“pass”/“fail”) relates to the reported example, but in general it is not required to be binary in the model (see also Figure 5.2) nor in the scheduling algorithms proposed hereafter. ................. 165

5.2 The DIAC skilled migrant visa processing workflow with an additional loop. ................................................................. 166

5.3 A Petri net with a single start place $p_s$ and a single end (sink) place $p_e$. Without the two dotted arcs $(p_3, t_1)$, $(t_4, p_2)$ the net is also a correct workflow net in the sense of Def. 5.7. A single token in $p_2$ enables both transition $t_3$ and $t_3$, but only one of them may fire. Thus $p_2$ corresponds to an XOR-split: $p_5$ is the related XOR-join. The branches following $p_1, p_2, p_3$ are AND-parallel branches, i.e., correspond to parallel branches in a DAG. Transitions $t_4, t_5$ are in a loop. The dotted arc $(t_4, p_2)$ makes the net not 1-safe (Def. 5.2), while $(p_3, t_1)$ may cause a deadlock. ........................................................................ 176

5.4 A Petri net with a set of conditional branches. ................................................. 185

5.5 A simple Petri net with a loop. ................................................................. 188

5.6 A Petri net subnet with overlapping loops. ..................................................... 191

5.7 Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function: (a) of the processor multiplier $p_{mul}$, given $|P| = \lceil vp_{mul} \rceil$; (b) of the multiplier $\lambda_{mul}$ of the critical arrival rate, given $\lambda = \lambda_{mul}\lambda_{max}$. Algorithms with variable greediness. ................. 206
List of Figures

5.8 (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation \( \text{nsd} = \frac{\sigma}{\mu} \) of the activity durations; (d) standard deviations of the normalized response time, as a function of \( \text{nsd} \). Algorithms with variable greediness. 207

5.9 Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function: (a) of the processor multiplier \( p_{\text{mul}} \), given \( |P| = \lfloor v p_{\text{mul}} \rfloor \); (b) of the multiplier \( \lambda_{\text{mul}} \) of the critical arrival rate, given \( \lambda = \lambda_{\text{mul}} \lambda_{\text{max}} \). Algorithms with variable greediness. 209

5.10 (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation \( \text{nsd} = \frac{\sigma}{\mu} \) of the activity durations; (d) standard deviations of the normalized response time, as a function of \( \text{nsd} \). Algorithms with parameter-driven greediness. 210

5.11 Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function: (a) of the processor multiplier \( p_{\text{mul}} \), given \( |P| = \lfloor v p_{\text{mul}} \rfloor \); (b) of the multiplier \( \lambda_{\text{mul}} \) of the critical arrival rate, given \( \lambda = \lambda_{\text{mul}} \lambda_{\text{max}} \). Algorithms with different job prioritizing and machine assignment heuristics. 211

5.12 (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation \( \text{nsd} = \frac{\sigma}{\mu} \) of the activity durations; (d) standard deviations of the normalized response time, as a function of \( \text{nsd} \). Algorithms with different greediness. 212

5.13 Sample mean and dispersion of the response time for an instance, normalized by the processing time of all the task realizations in the instance’s critical path, as a function: (a) of the processor multiplier \( p_{\text{mul}} \), given \( |P| = \lfloor v p_{\text{mul}} \rfloor \); (b) of the multiplier \( \lambda_{\text{mul}} \) of the critical arrival rate, given \( \lambda = \lambda_{\text{mul}} \lambda_{\text{max}} \). Algorithms with different greediness. 214

5.14 (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation \( \text{nsd} = \frac{\sigma}{\mu} \) of the activity durations; (d) standard deviations of the normalized response time, as a function of \( \text{nsd} \). Algorithms with different greediness. 215
List of Figures

5.15 Sample mean and dispersion of the response time for an instance, normalized by the processing time of all the task realizations in the instance’s critical path, as a function: (a) of the processor multiplier $p_{\text{mul}}$, given $|P| = |vp_{\text{mul}}|$; (b) of the multiplier $\lambda_{\text{mul}}$ of the critical arrival rate, given $\lambda = \lambda_{\text{mul}}\lambda_{\text{max}}$. Algorithms with parameter-driven greediness. ........................................... 217

5.16 (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $nsd = \frac{\sigma}{\mu}$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $nsd$. Algorithms with parameter-driven greediness. ........................................... 218

5.17 Sample mean and dispersion of the response time for an instance, normalized by the processing time of all the task realizations in the instance’s critical path, as a function: (a) of the processor multiplier $p_{\text{mul}}$, given $|P| = |vp_{\text{mul}}|$; (b) of the multiplier $\lambda_{\text{mul}}$ of the critical arrival rate, given $\lambda = \lambda_{\text{mul}}\lambda_{\text{max}}$. Algorithms with different job prioritizing and machine assignment heuristics. ........................................... 220

5.18 (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $nsd = \frac{\sigma}{\mu}$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $nsd$. Algorithms with different job prioritizing and machine assignment heuristics. ........................................... 221

5.19 Sample mean and dispersion of the response time for an instance, normalized by the processing time of all the task realizations in the instance’s critical path, as a function: (a) of the processor multiplier $p_{\text{mul}}$, given $|P| = |vp_{\text{mul}}|$; (b) of the multiplier $\lambda_{\text{mul}}$ of the critical arrival rate, given $\lambda = \lambda_{\text{mul}}\lambda_{\text{max}}$. Algorithms with different job prioritizing and machine assignment heuristics. ........................................... 222

5.20 (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $nsd = \frac{\sigma}{\mu}$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $nsd$. Algorithms with different job prioritizing and machine assignment heuristics. ........................................... 223
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Taxonomy in Scheduling for Parallel Processing: field $\alpha$</td>
<td>44</td>
</tr>
<tr>
<td>2.2</td>
<td>Taxonomy in Scheduling for Parallel Processing: field $\beta$</td>
<td>45</td>
</tr>
<tr>
<td>2.3</td>
<td>Taxonomy in Scheduling for Parallel Processing: field $\gamma$</td>
<td>45</td>
</tr>
<tr>
<td>2.4</td>
<td>Kendall’s notation with Lee’s and Taha’s extensions: selected values</td>
<td>70</td>
</tr>
<tr>
<td>3.1</td>
<td>Outline of the experiments for the $P</td>
<td>Q</td>
</tr>
<tr>
<td>3.2</td>
<td>Statistical significance of the performance difference between TOFF* and TOFF-E=0.01. Two-tailed Mann-Whitney $U$ test: sample size $5 \cdot 10^4$; * = significance at $p &lt; 0.05$, ** = significance at $p &lt; 0.01$, *** = significance at $p &lt; 0.001$.</td>
<td>107</td>
</tr>
<tr>
<td>3.3</td>
<td>Statistical significance of the performance difference between TOFF*/NONORM and TOFF*/MAX. Two-tailed Mann-Whitney $U$ test: sample size $5 \cdot 10^4$; * = significance at $p &lt; 0.05$, ** = significance at $p &lt; 0.01$, *** = significance at $p &lt; 0.001$.</td>
<td>115</td>
</tr>
<tr>
<td>4.1</td>
<td>Outline of the experiments for the $G</td>
<td>G/c : P</td>
</tr>
<tr>
<td>Table No.</td>
<td>Description</td>
<td>Sample Size</td>
</tr>
<tr>
<td>-----------</td>
<td>-------------</td>
<td>-------------</td>
</tr>
<tr>
<td>4.2</td>
<td>Statistical significance of the performance difference between OBN and GDY-P. Two-tailed Mann-Whitney $U$ test: sample size $5 \cdot 10^4$; $^* = \text{significance at } p &lt; 0.05$, $^{<strong>} = \text{significance at } p &lt; 0.01$, $^{</strong>*} = \text{significance at } p &lt; 0.001$.</td>
<td>$5 \cdot 10^4$</td>
</tr>
<tr>
<td>4.3</td>
<td>Statistical significance of the performance difference between OBN-CM and OBN-MSD. Two-tailed Mann-Whitney $U$ test: sample size $5 \cdot 10^4$; $^* = \text{significance at } p &lt; 0.05$, $^{<strong>} = \text{significance at } p &lt; 0.01$, $^{</strong>*} = \text{significance at } p &lt; 0.001$.</td>
<td>$5 \cdot 10^4$</td>
</tr>
<tr>
<td>4.4</td>
<td>Statistical significance of the performance difference between OBN and TOFF-NOOBN. Two-tailed Mann-Whitney $U$ test: sample size $5 \cdot 10^4$; $^* = \text{significance at } p &lt; 0.05$, $^{<strong>} = \text{significance at } p &lt; 0.01$, $^{</strong>*} = \text{significance at } p &lt; 0.001$.</td>
<td>$5 \cdot 10^4$</td>
</tr>
<tr>
<td>5.1</td>
<td>Outline of the experiments for $G/G/c: P/Q</td>
<td>v_i, c_{ij} \sim stoch, wn</td>
</tr>
<tr>
<td>5.2</td>
<td>Outline of the experiments for $G/G/c: P/Q</td>
<td>v_i, c_{ij} \sim stoch, wn</td>
</tr>
<tr>
<td>5.3</td>
<td>Statistical significance of the performance difference between TOFF/wXOR and TOFF/mXOR. Two-tailed Mann-Whitney $U$ test: sample size $10^4$; $^* = \text{significance at } p &lt; 0.05$, $^{**} = \text{significance at } p &lt; 0.01$.</td>
<td>$10^4$</td>
</tr>
<tr>
<td>5.4</td>
<td>Statistical significance of the performance difference between TOFF/wXOR and DLS/mXOR. Two-tailed Mann-Whitney $U$ test: sample size $10^4$; $^* = \text{significance at } p &lt; 0.05$, $^{**} = \text{significance at } p &lt; 0.01$.</td>
<td>$10^4$</td>
</tr>
<tr>
<td>5.5</td>
<td>Statistical significance of the performance difference between TOFF/vseq and TOFF/sum. Two-tailed Mann-Whitney $U$ test: sample size $10^4$; $^* = \text{significance at } p &lt; 0.05$, $^{**} = \text{significance at } p &lt; 0.01$.</td>
<td>$10^4$</td>
</tr>
<tr>
<td>5.6</td>
<td>Statistical significance of the performance difference between TOFF/vseq and DLS/vseq. Two-tailed Mann-Whitney $U$ test: sample size $10^4$; $^* = \text{significance at } p &lt; 0.05$, $^{**} = \text{significance at } p &lt; 0.01$.</td>
<td>$10^4$</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

A workflow is a collection of coordinated tasks designed to carry out a well-defined complex process [174]. The tasks reflect atomic business, production or computational activities. Some workflows explicitly capture inter-task communication, which corresponds to the transfer of intermediate products or interim computation data. Tasks and acts of inter-task communication shall be referred to with the umbrella term activities.

The requirement that the collection of activities be coordinated corresponds to the fact that a workflow must specify a set of relationships between them. This is true both of the concrete workflow (business process or application) and the workflow model (schema). The schema, in essence, is a blueprint for multiple concrete workflows that share some constraints, which may be of various nature: It specifies what kind of relationships exist between all the possible realizations of the activities in the model.

The set of relationships can be viewed from different perspectives, or aspects. Jablon- ski and Bussler [117] individuated five such perspectives. The functional aspect defines what a workflow does. The behavioral aspect specifies how this is achieved in terms of execution order and dependencies, i.e., constraints in the time domain. In other words, it describes the control flow. The informational perspective relates to data dependencies and sharing. The organisational aspect specifies resource allocation, answering to the question who performs which activities using which means. The operational perspective contains the description of the interaction of the workflow with its environment, e.g., receiving external inputs and outputs. Van der Aalst ([243], [247]), on the other hand, individuated three dimensions of a workflow. The case dimension is related to the distinction of multiple concrete workflows which follow the same schema but
Chapter 1. Introduction

are independent. In other words, workflows are enacted (executed) giving rise to cases or workflow instances. These are the actual business processes that perform a specific set of tasks, i.e., achieve a complex task on behalf on a beneficiary (consumer). The process dimension describes the relationships of the tasks and the workflow’s routing order: It corresponds directly to Jablonski and Bussler’s behavioral aspect. The resource dimension isolates the concerns pertaining, on the lowest level, to the actuators of the specific tasks and other resources needed to perform them. On a higher level, it pertains to the roles and organizational units that group such resources. A concrete system designed to execute a workflow in accordance with its resource requirements is termed a workflow engine (enactment system). Based on their previous work, van der Aalst et al. [250] formalized the different relationships in a set of workflow patterns. They individuated twenty control flow patterns, later extended by Russell, Ter Hofstede and Mulyar [203] by another twenty-three patterns. Taking into consideration Jablonski and Bussler’s informational perspective [117], Russell et al. [202] categorized thirty-nine data flow patterns. The same authors [204] extended this research by adding forty-three resource patterns. There exist a large number of modeling languages and tools [182] that cover a varying number of patterns. For example, approaches based on colored Petri nets, such as Yet Another Workflow Language (YAWL, [249]), were devised with the specific aim to enable the modelling of all possible patterns belonging to all perspectives, and to achieve this goal while preserving formal rigor, consistency and minimal syntax, lacking in most comparable business-domain efforts ([251], [255], [268]).

The relationships implied by the control flow dimension can be expressed by a set of precedence constraints or by a process model. These can be of varying expressive power, i.e., the range of control flow constructs that they can capture. For example, the classical directed acyclic graph (DAG) model can express a control flow with series-parallel execution paths or, alternatively, partially ordered sets of activities [23]. In this model, two activities that may be executed in parallel must ultimately both be executed: Hence the term AND-type precedence constraint is sometimes employed to refer to the constraints expressible in this model. AND-OR-graphs ([61], [94]) can also express OR-type constraints that represent choices (alternative flow paths). In addition to AND- and OR-constraints, Petri nets ([244], [210]), process algebras [195] and other models of concurrency ([18], [58], [59], [264]) can also represent an iterative control flow, i.e., repeated execution of a subset of activities, based on conditions evaluated at
Based on scenarios from multiple domains, including business and parallel computing, this thesis argues that realistic applications require increasingly expressive control flow models. In particular, there are cases in which conditional and iterative flow must be represented explicitly. Accordingly, the expressive power of the models considered grows gradually throughout the thesis. The starting point is a single instance of a directed acyclic graph, whose nodes correspond to tasks and edges represent inter-task communication. In the next step, the case dimension of the workflow is captured through a stochastic arrival process which gives rise to a queue of workflow instances. The arrival is characterized by an arbitrary distribution, with a known average arrival rate. The final and most expressive application model considered admits multiple instances of a workflow net ([244], [254]). The workflow net is a subclass of Petri nets, a digraph with two sets of objects: places and transitions, connected by an alternating flow relation (place-to-transition and transition-to-place arcs). As it is common in the workflow context, the transitions are assumed to represent tasks and the places their pre- and postconditions. We additionally map the inter-task communications to the transition-place arcs.

The execution time of each task and the length of each communication delay are associated with real-valued variables, which describe their nominal run time. The nominal, or intrinsic time is the run time on a processing unit or communication channel of unitary speed. This thesis takes into consideration workflow enactment systems with homogeneous and uniformly heterogeneous resources, corresponding to processing units and communication channels. In uniformly heterogeneous environments, processors and channels have known, constant speed factors with respect to the nominal activity run times. These nominal activity run times are assumed to be described by stochastic variables with known, arbitrary distributions, which must satisfy only mild requirements. Notably, most comparable approaches that admit stochastic activities consider exclusively specific distribution families (e.g., stochastic scheduling [179]).

For an arbitrary workflow schema in compliance with each model, this thesis provides efficient policies which optimize the execution of a set of instances of that schema. The process in which a policy makes decisions to solve this problem is referred to as scheduling and any solution is a schedule. In our case, optimality implies minimization of selected objectives. The focus remains on time-domain objectives that are of primary
importance from the point of view of the consumer, on whose behalf the workflow is
executed by a workflow engine comprised of a number of resources. When users
submit requests, their main concern is the perceived response time of the system. It
coincides with the execution time in a problem with a single workflow instance, or with
the sum of waiting and execution time, if the system is shared with other customers
and their workflow instances.

1.1 Research Questions

This thesis provides answers to the following research questions related to the assign-
ment of resources to tasks of a number of workflow instances in stochastic environ-
ments.

**Offline Scheduling of DAG-reducible Stochastic Workflows**

1. Given that the general deterministic schedule length minimization is an NP-hard
combinatorial problem, how should an additional uncertainty in the problem in-
puts, i.e., activity run times, be managed by an offline scheduling policy?

2. As the problem of computing the distribution of the schedule length is #P-
complete and given that optimizing a schedule may require the evaluation of
many partial solutions, is it possible to devise methods which can estimate partial
schedules efficiently? How can the specific features of the optimization problem,
e.g., the addition and comparison operators employed in the objective function,
guide the choice of the estimation methods?

3. If a task is considered for insertion into an idle time slot on a processor and its
tentative start and finish times and those of the previously scheduled tasks are
stochastic variables, what is a computationally efficient method of estimating
the impact of a scheduling decision on the whole solution?

4. Given that the run time of an activity that has not been scheduled yet will depend
on the speed of the resource (processor or communication channel) it will be
scheduled to, what is the best strategy of estimating the resource speed before
the choice is made?
1.1. Research Questions

5. In platforms with different processing capacity, i.e., different speed and number of processing units, how meaningful is the ratio of the available processing units to the size of the task graph and how can it be employed to guide the behavior of the scheduler?

6. For all candidate approaches and their component heuristics, i.e., partial schedule and resource speed estimation strategies, what is the performance in platforms with different processing capacity? What is the performance under different computation-to-communication ratios? How robust are the resulting schedules in settings under different levels of stochastic uncertainty?

Online Scheduling of DAG-reducible Stochastic Workflows

1. How should the uncertainty in the problem inputs be managed by an online scheduling policy? In particular, given a task considered for scheduling on a processor, if its tentative finish time and the tentative finish times of other tasks of different, possibly higher priority, are stochastic variables, which methods can estimate the impact of a scheduling decision efficiently?

2. What methods can improve the accuracy of the estimation of partial schedules in an online scheduler? How can the specific features of the optimization problem guide the choice of these estimation methods?

3. Given the expected response time of the system as a global objective, how should a scheduling policy balance the objectives of expected instance makespan and in-queue waiting time, which may be conflicting for different instances?

4. Given a stochastic arrival process of workflow instances, what is a meaningful measure of the different load conditions of the system, resulting from different arrival rates of new workflow instances?

5. If a specific arrival rate is determined with a reference scheduling policy, how can the ratio of the actual arrival rate to the reference rate guide the behavior of the scheduler?

6. What is the performance of the candidate scheduling policies in platforms with different processing capacities? How robust are the schedules determined by these policies in settings with different levels of stochastic uncertainty?
Scheduling of Petri Net-reducible Stochastic Workflows

1. What features of the application model are necessary for realistic workflows that are unavailable in the directed acyclic graph model? Which candidate models have the required expressive power and which criteria should guide the choice of the model?

2. If the control flow admits conditional execution and iterations of activities, which efficient methods can be employed to determine the set of schedulable jobs in a policy that is required to make many such determinations?

3. How can the task priority determination problem be solved in the presence of conditional branches and loops of activities in a setting with stochastic activities? Which methods can support conditional paths corresponding to both exclusive and non-exclusive choices in the control flow?

4. What is an efficient strategy for the problem of estimating the length of a loop containing stochastic activities? How accurate can such a strategy be? Which methods can estimate the lengths of finite and potentially infinite loops?

5. Finally, how can an overall set of length estimation strategies be designed to support combinations of series-parallel, conditional and iterative execution paths?

6. What is the performance and robustness of the proposed approach in settings with different load conditions and different levels of stochastic uncertainty?

The above research questions will be investigated in detail in Chapters 3-5 and answered in a summarizing form in Chapter 6. The performance and robustness of the proposed approaches will be shown in detail in the respective chapters.

1.2 Contributions

Offline Scheduling of DAG-reducible Stochastic Workflows

1. A novel and efficient policy for the problem of minimizing the expected makespan when a single DAG-structured workflow instance with stochastic activities is
1.2. Contributions

scheduled to either a homogeneous or uniformly heterogeneous platform. The policy is based on an original trade-off mechanism between the gain, associated with insertion-based (greedy) scheduling of jobs of lower priority, and the loss, represented by the stochastic delay that such greedy scheduling may induce for higher priority jobs. The trade-off mechanism represents thus a parameter-driven approach to greediness. The specific value of the driving parameter is determined based on the relative speed of the scheduling platform with respect to the size of the task graph.

2. A novel method of improving the accuracy of the estimates of the values of the stochastic variables associated with the workflow’s activities, based on the employment of partial moments of the relevant probability distributions.

Online Scheduling of DAG-reducible Stochastic Workflows

1. A general online scheduling policy for the problem of minimizing the expected response time, when a queue of DAG-structured workflow instances, each with stochastic activities, is scheduled onto either a homogeneous or uniformly heterogeneous platform. The policy is parametrized, so that different (online and offline) job priority determination strategies and estimates of the stochastic variables of interest can be plugged in. The configurable trade-off mechanism driving the choice between greedy and lazy scheduling, put forward for the offline scheduling problem, is adapted to the online scheduling setting and parametrized with respect to the rate of the arrival process characteristic of the queue.

2. A method of improving the accuracy of the estimates of stochastic variables, based on the employment of conditional partial moments of the probability distributions for the determination of the duration of currently running activities, which results in an improvement of the expected makespan sub-objective at the cost of an adjustable, linear increase in complexity.

3. A local look-ahead subroutine to minimize the instance waiting time sub-objective by allowing incoming workflow instances to start before the completion of the instances under processing, when said completion is unlikely to be affected.

Scheduling of Petri Net-reducible Stochastic Workflows
Chapter 1. Introduction

1. A scheduling policy for the problem of minimization of the expected response time, when a queue of instances of a workflow net with stochastic activities is scheduled onto a homogeneous or uniformly heterogeneous platform. The policy is configurable and it enables the employment of different strategies of estimating lengths of conditional path sets and loops of activities, in addition to different stochastic variable value estimation methods and job priority determination heuristics.

2. A computationally efficient method of determining the set of executable jobs, based on labelling of inactive jobs and propagated elimination of jobs whose parents are labelled inactive.

3. An efficient method of estimating the length of conditional paths involving stochastic activities, which supports path sets corresponding to both exclusive and non-exclusive choices.

4. An efficient method of estimating the length of loops containing stochastic activities, which supports arbitrary combinations of loops with a finite or potentially infinite number of iterations and provides more accurate estimation methods for loops containing currently running activities.

The performance of the proposed approaches is tested against comparable state-of-the-art algorithms known from the subject literature. Where applicable, these algorithms are adapted to work in settings with multiple workflow instances and augmented with mechanisms required to support more expressive application models, in particular workflow nets. Reference makespan-centric algorithms are also extended and tested with different mechanisms that can be employed to minimize the total response time of the system.

1.3 Thesis Overview

The thesis is organized in six chapters.

Chapter 2 presents a contextual view of the thesis and introduces basic notions and taxonomical classifications of related problems from operations research in general, and deterministic and stochastic scheduling in particular.
Essential definitions of scheduling framework, platform, application models, schedules and optimality are provided. Fundamental related work and results that are relevant for all approaches presented throughout the thesis are reported.

Chapter 3 investigates the problem of scheduling a workflow containing stochastic activities, with the objective of minimizing the expected schedule length. The scenario features a workflow with an arbitrary directed acyclic graph structure and requires a static schedule, known in advance.

An offline policy is proposed to solve this problem. It has a polynomial time complexity with respect to the size of the inputs, thus is computationally efficient.

The performance and the robustness of the policy are evaluated experimentally in a competitive analysis procedure.

Chapter 4 investigates the problem of scheduling multiple instances of a workflow with an arbitrary directed acyclic graph structure containing stochastic activities. The instances are assumed to arrive into the system in accordance to an arbitrary stochastic process, with a known average arrival rate. The global objective is to minimize the expected response time of the system, which corresponds to the expectation of the sum of schedule length and in-queue waiting time. This scenario admits real-time monitoring and dispatching of tasks, thus scheduling decisions can be made at runtime.

Under the assumptions outlined, an online scheduling policy is proposed to solve this problem. It has a polynomial time complexity with respect to the inputs.

Experimental evaluation of the performance and robustness of the proposed policy in different problem settings concludes the chapter.

Chapter 5 investigates scheduling problems that can be formulated, when the underlying workflow model admits conditional and iterative execution of stochastic activities. In particular, the Petri net-based workflow net model is assumed as the application model. A queue of workflow instances conforming to this model is generated by a stochastic arrival process with a known average rate. The expected response time of the system must be minimized. Online monitoring and task dispatching are allowed in the scenario.

An online scheduling policy is proposed to solve this problem. It has a polynomial time complexity with respect to most inputs. The complexity is worst-case exponential
in the case of conditional paths within the workflow, corresponding to non-exclusive choices of activities to be executed. A less accurate path length estimation strategy for the policy is provided, which reduces the complexity to a polynomial one, if the number of non-exclusive choices is excessive.

Experimental evaluation of the performance and robustness of the proposed policy with different path length estimation strategies concludes the chapter.

Finally, Chapter 6 draws conclusions, summarizes the answers provided for the research questions outlined in the introduction. It also opens an outlook to new, related issues and challenges and to possible ways of facing them.
Publications

The research presented in this thesis has led to the following publications:


Chapter 2

Background and Preliminaries

In this Chapter, a contextual view on the problematic presented in this thesis is provided.

The basic concepts of job, application and application model are presented, as well as the definitions of renewable and non-renewable resources and scheduling platforms. The general notion of schedule and scheduling problem is introduced, followed by those of scheduling policies and their performance with respect to a set of objectives. Next, the specific relevant schedule optimization problems are presented, accompanied by the respective complexity results and state-of-the-art methods applied to finding and evaluating their solutions. A particular attention is dedicated, in this respect, to makespan and response time optimization issues in stochastic problem settings. In addition to the basic definitions and theorems, the notation used throughout this thesis is established.

2.1 Problem Context

Gue and Thomas [92] proposed a classification of decision problems in operations research, namely: allocation, scheduling, sequencing and waiting lines, as well as competition, inventory, replacement and search problems.

An allocation problem is defined as the allocation (assignment) of usually limited resources to a number of entities which require them for an action. Such an action may be the production of consumer goods, that may require a temporary, non-destructive
2.2. Scheduling Framework

hold on those resources, such as the use of a machine, or their consumption, e.g., the consumption of raw material. Allocation problems are of interest of the work presented herein only in relation to scheduling.

Scheduling can be understood in a strict sense (as in [92]) or in a broad one (see end of this section). A scheduling problem in the strict sense is the timing of arrival and departure of units of work requiring processing or service, a sequencing problem concerns the order of the processing. In practice, the class of resource-unconstrained project “scheduling” problems is the only one containing purely sequencing problems and is of limited interest for this work.

Finally, a waiting line problem relates to a randomly variable demand for services and the planning of facilities to meet this type of demand. Both processing order (queue discipline) and waiting line problems are at the center of research in queuing theory (see Section 2.18).

Scheduling in general, however, implies allocation and sequencing, while scheduling jobs arriving at random times into any processing system also implies managing waiting lines. In a similar manner, this thesis and the related work is concerned with a set of problem classes that combine allocation, scheduling, sequencing and, unless otherwise indicated (see Chapter 3), waiting line problems. All of the aforementioned problems will be referred to using the umbrella term “scheduling”, unless a more detailed specification is required.

2.2 Scheduling Framework

A scheduling framework is composed of an application model and a scheduling platform.

A resource is a quantifiable entity behaving according to two basic models:

- a renewable resource may be locked at the start of an activity and released at its completion, such as a machine or human operator performing an activity

- a non-renewable resource may be consumed in the process of the execution of an activity, such as time, energy, raw materials.
Chapter 2. Background and Preliminaries

An alternative, but conceptually equivalent, set of definitions is provided by Brucker et al. [39]. According to their formulation, “renewable” refers to a number of units of a resource available for every interval of the planning horizon, whereas “non-renewable” to a number of units of a resource available for the entire planning horizon. Also of some relevance within the mentioned framework are the so-called double-constrained resources. The distinction is due to the fact that any resources may be limited within both the horizon of the whole project and smaller time periods (e.g., a budget, defined both for the entire makespan of a project and per each month of its lifespan).

These definitions are made in the context of Resource-Constrained Project Scheduling (RCPS, see, e.g., Herroelen, Demeulemeester and De Reyck [102]), which is a problem that generalizes the scheduling problems presented in this thesis with respect to the admissible and required resource sets. RCPS approaches, however, mostly assume deterministic resource requirements expressed by natural numbers and discrete time periods, which is not the case in this thesis. Many other, specialized resource models have been taxonomically described and catalogued by Hartmann and Briskorn [99], including those specific to multi-mode RCPS (MRCPS), a generalization of RCPS introduced by Elmaghraby in [67].

2.2.1 Basic Application Model

Let \( \rho_t \) be a non-renewable resource type and \( w(\rho_t) \) be a quantity associated with it, with a value in \( \mathbb{R}^+ \), that may be a constant known a priori (i.e., have a deterministic value), or a variable with a known range or probability distribution, or a value unknown until the occurrence of an event. We shall write \( w(\rho_t, a) \) if the quantity of the resource intended is also associated with an activity \( a \).

**Definition 2.1.** A job \( j \) is an activity, atomic or structured, that may be associated with a required processing (execution) time, or intrinsic weight, which is a non-renewable resource in the above sense. In particular, the value of \( w(\rho_t, j) = w(j) \) may be given by a constant, a range, a probability distribution known a priori, or be unknown until the job’s instantiation, its assignment to a machine, or its completion.

A job \( j \) may have a release date \( r(j) \in \mathbb{R}^+ \cup \{0\} \), that indicates an arbitrary time, which is the earliest allowed start time for the job.

Throughout this thesis \( w(j) \) shall be considered as a random variable with a known
2.2. Scheduling Framework

probability distribution (or degenerate distribution, i.e., assuming deterministic values), that is intrinsic to the job \( j \), i.e., independent of the assignment of the job to a machine, or the moment of instantiation. In other words, this is the proper (nominal) processing time of a job, that, with respect to all possible machine assignments, may be regarded as the processing time normalized across these assignments. This does not preclude the existence of different processing times distributions for the same job, when assigned to machines of various speeds, see Sect. 2.2.2.

**Definition 2.2.** A precedence constraint between two jobs \( j_i, j_k \), indicated by \( j_i \prec j_k \), or \( e_{ik} \), means that job \( j_i \) must be completed before job \( j_k \) can be started. The definition of a set of precedence constraints \( E \) can also be formulated using job indexes, as follows: \( E = \{ e : e = (i, k), i \neq k \} \) for some \( i, k \).

An application model with precedence-constrained jobs is representable in the form of a directed graph \( G = (J, E) \), where \( J \) is the set of nodes (jobs) and \( E \) the set of edges (constraints) connecting them.

Traditionally the graph (or, alternatively, the set of the precedence constraints) is required to be acyclic, i.e., to satisfy the condition in Def. 2.3.

**Definition 2.3.** The set of precedence constraints and the graph describing them are termed acyclic if and only if the following is true: \( \forall i, k, m : e_{ik}, e_{km} \in E \Rightarrow e_{mi} \notin E \).

Almost all scheduling approaches in the subject literature consider only directed acyclic graphs (DAGs) as application models. Some of them impose further restrictions on the structure of such graphs. Sect. 2.16.2 provides details on the most notable heuristic approaches.

In practical application models, a precedence constraint (or, equivalently, an edge in the corresponding graph) may be associated with a communication link between connected jobs.

**Definition 2.4.** A communication time (delay) between two jobs \( j_i, j_k \), indicated as \( d(j_i, j_k) = d_{ik} \in \mathbb{R}_+ \cup \{0\} \) is the time, that may be known with any degree of certainty.

\(^1\)The symbols \( d(j_i, j_k), d_{ik} \), with a lower-case “d” for delay, will be used throughout this thesis, instead of \( c(j_i, j_k), c_{ik} \) (lower-case “c” for communication), usually found in scheduling literature, to any avoid confusion with completion time of job \( j \) (\( C_j \), upper-case “C”) and total application completion time (makespan, or schedule length \( C_{max} \)).
needed to transfer working data between these jobs. A communication link implies the existence of a precedence constraint: $\exists d_{ik} \Rightarrow \exists e_{ik}$.

Let $J$ be a set of jobs and $E$ the corresponding set of precedence constraints. Similarly to the processing times and release dates of the single jobs in $J$, the sets $J, E$ may be known a priori or become known during the execution of the application.

In all application models considered in this thesis, a precedence constraint may be represented as an edge in a task graph. In this case the communication time $d_{ik}$ represents the intrinsic (i.e., assignment-independent) weight of the edge, analogously to the observation in Def. 2.1: $d_{ik} = w(e_{ik})$.

**Definition 2.5.** An application is a tuple $(J, E)$, where $J = \{j_1, \ldots, j_k, \ldots, j_n\}$ is a set of $|J| = n \in \mathbb{N}_+$ jobs and $E \subset J \times J$ is a set of precedence constraints.

Like the processing times and release dates of the jobs in $J$, the set of jobs $J$ itself may be known a priori or become known during the execution of the application.

Conventionally, the terms “stochastic setting” and “stochastic graph” shall be used henceforth to imply that at least one activity in the graph is associated with a stochastic quantity of a (non-renewable) resource consumed in the process.

### 2.2.2 Scheduling Platform

Let $\rho_p$ be a renewable resource type. Let a generic machine, or processor be a resource of type $\rho_p$, and let the quantity (multiplicity) of that resource be a known, constant value in $\mathbb{N}_+$.

**Definition 2.6.** A scheduling platform consists of a set $P = \{p_1, \ldots, p_l, \ldots, p_m\}$ of $|P| = m$ machines, or processors, available in parallel in a system targeted at allocating and processing a set of applications, and of a set $H \in P \times P$ of physical communication channels connecting these machines.

The machines can be organized in different topologies or architectures, being fully or arbitrarily connected by means of the communication channels. The absence of a physical channel between two processors $p_l, p_q$ implies that two jobs $j_1, j_k$ allocated
one to each of them may not communicate directly, even when required by the application model. Different machines can also have varying features, in particular, different speeds (or costs, reliabilities, etc.) of processing the same jobs.

The effective processing time of a job $j$ can be different from the machine-independent processing time as per Definition 2.1. In general, the processing time of a job $j$, if known, is given by a real-valued (deterministic or stochastic) variable $w(j, p)$.

If the processing times are known, at least in terms of probability distributions, the following customary distinctions can be made and definitions put forward.

**Definition 2.7.** If the processing times are known and if $w(j, p) = w(j, q) = w_j$ and $w_j$ is a known constant for all $p, q$, the processors are identical. If there exists a single constant $k \in \mathbb{R}^+$ such that $w(j, p) = kw(j, q)$ for all $j$ and any pair $p, q$, the processors are uniform. If no such single constant exists, the processors are unrelated. Identical, uniform and unrelated channels are defined analogously. A scheduling platform with identical processors and identical communication channels forms a homogeneous processor environment. A platform in which either the processors or the communication channels are uniform but not identical is termed uniformly heterogeneous.

Scheduling platforms with unrelated processors or channels (non-uniformly heterogeneous platforms) are not investigated in this thesis.

A generic additive cost model taking into account computation and communication costs in arbitrarily connected application platforms was first introduced by Chu et al. [52] and expanded by Ma, Lee and Tsuchiya [161], who additionally consider constraints related to task attributes other than costs (memory requirements, task conflicts, redundancies, etc.). The latter work also shows how the topology and the parametrizations for application platforms with uniform and unrelated processors can be derived by using a network preprocessor. For a taxonomical classification of the parallel processing environments, refer to Sect. 2.12.

In particular problem settings, some of the processors in $P$ can also be dedicated to specific jobs, meaning that no other processor is meant or able to perform these specific jobs, i.e., there exist some subset of jobs $V_*$ such that $J_* = \{j \in J : \exists P_* \in P : p \notin P_* \Rightarrow w(j, p) = \infty\}$. 
2.2.2.1 Interdependency Issues of Platform and Job Models

All the previous definitions tacitly assume that a job may only be executed by one machine throughout the entirety of its runtime. Problem settings with moldable and malleable jobs drop this assumption.

**Definition 2.8.** A moldable job is a job that may be executed in parallel by a set of machines that may not change during execution.

**Definition 2.9.** A malleable job is a (moldable) job that may be executed in parallel by a set of machines that may change during execution.

Note that it is a necessary but not sufficient condition for a job to be moldable, if it is malleable. In the light of this and for the sake of completeness, the following Definition can also be formulated by exclusion.

**Definition 2.10.** A rigid job is a job that is not moldable.

In other words, for a moldable or malleable job, in general, it is possible to assign it to more than one machine simultaneously and the processing time depends on this assignment: in particular, it is usually assumed to be a non-increasing function of the number of machines executing the job, with more restrictive assumptions (e.g., quasi-linearity) in many practical settings.

It is easy to prove that a moldable job having an intrinsic processing time (weight) of $w$ that may be allocated to a maximum of $m$ identical machines and exhibits a linear speed-up (see Definition 2.25) is equivalent to a set of $m$ rigid parallel jobs of weight $w/m$. This reasoning does not change if the machines are uniform. Realistic models with moldable or malleable jobs, however, usually assume a non-linear speed-up with respect to the number of machines. Throughout this thesis, only rigid jobs are explicitly considered (and, by equivalence, moldable jobs with linear speed-ups).

2.3 Schedule

With an application model and a scheduling framework, it is possible to define a schedule as follows.
2.3. Schedule

**Definition 2.11.** Given a set of jobs $J$ and a set of processors $P$, a schedule $S$ is an allocation of jobs in $J$ to (a subset of) $P$ and to a subset of time intervals on these processors.

Formally, let $p(j) \in P$ be the processor assigned to $j$ and let $S(j,p) = S_j, C(j,p) = C_j \in \mathbb{R}^+ \cup \{0\}$ be the start and finish time of a job $j$ on $p(j)$. Then a schedule is a tuple $S = (\{S_j\}, \{C_j\}, \{p(j)\})$.

A schedule is complete if all the jobs are assigned, otherwise it is partial.

A schedule is usually required to be *feasible*, which informally means that the execution plan implied by the schedule is practicable. In other words, a schedule is feasible if and only if all the precedence constraints, communication delays and release dates between jobs, if applicable, are respected and no two jobs overlap on any processor.

**Definition 2.12.** A schedule $S$ is feasible if and only if it satisfies the following conditions:

- release dates are respected: $S_j \geq r_j$
- precedence constraints are observed and communication delays are considered, if applicable, i.e., a job must not start on a given processor earlier than all of its parent jobs have been completed and, if necessary, the communication has arrived from the processors on which the parents have been executed, formally: $\forall (i,j) \in E \Rightarrow S_j \geq C_i + d_{ij} 1_{P \setminus \{p(i)\}}(p(j))$
- job assignments do not overlap on any single processor: $p(j) = p(k) \Rightarrow S_j \geq C_k \lor S_k \geq C_j$ for all $j, k \in J, j \neq k$ and for all $p \in P$.

A schedule is additionally complete, if each job has (at least) one processor assigned to it: $\forall j \in J \Rightarrow \exists p(j) \in P$ ($J \rightarrow P$ is surjective).

In the above definition, the classical assumption is followed that a communication delay is only induced when an actual transfer of data between two separate processing units is required, depending on the scheduling decisions taken. Formally, in this case, $1_{P \setminus \{p(i)\}}$ is a membership function, for which the following definition holds.
Chapter 2. Background and Preliminaries

Definition 2.13. The membership function \(1_{P \setminus \{p(i)\}}\) is given by:

\[
1_{P \setminus \{p(i)\}}(p(j)) = \begin{cases} 
1 & \iff p(j) = p(i) \\
0 & \text{otherwise}
\end{cases}
\] (2.1)

A communication delay as per the above definition is optional in the sense that there are schedules in which it is not enforced. As noted, it always implies a precedence constraint (clearly, the converse is not true) and in fact it generalizes the notion of a precedence constraint, by assigning it a real-valued function (or probability distribution).

Additionally, reasonable schedules are required to avoid total deliberate idle times, i.e., a situation in which no job is under processing (for the very straightforward definition see Radermacher [196]).

In general, it is admissible for a schedule to assign more than one processor to a job. Under certain assumptions on the application model or the nature of the tasks for particular objective functions it is advantageous to do so.

A schedule is preemptive if it admits interruption and resumption of a job, otherwise it is non-preemptive. In a non-preemptive schedule, a job may only be started if a previously commenced job has been completed, i.e., it formally holds: \(\forall j \neq k \in J \implies S_k \geq C_j \lor S_j \geq C_k\). The schedules examined in this thesis are non-preemptive.

Summarizing, a generic scheduling problem \(P\) is a problem of finding solution(s) as per Def. 2.11. The solutions must be feasible (2.12) and may optionally satisfy a set of constraints and/or other criteria (e.g., optimality with respect to some measure, see Def. 2.22). Given the previous definitions, it is clear that the set of jobs \(J(P)\) is uniquely defined by \(P\).

2.3.1 Equivalences

Release dates are additional constraints imposed on the start times of jobs in a schedule.

Theorem 1. Release dates can be equivalently expressed by precedence constraints with communication links, given a dedicated processor and a feasible schedule.

Proof. Let \(J = \{j_i\}\) with \(i \in \mathbb{N}_+\) be a set of jobs, let \(j_k\) be any job from the set \(J\) and
2.3. Schedule

Let \( r(j_k) \in R_+ \cup \{0\} \) be its release date. By definition of release dates, the job \( j_k \) can be started at or after \( r(j_k) \).

Let \( j_0 \) be an artificially introduced job, such that \( j_0 \) is the single entry node of the task graph and a parent of all the jobs in \( J \). Let \( j_0 \) be also the parent of the job \( j_k \) and let this precedence constraint be associated with a communication delay \( d(j_0, j_k) \). In any feasible schedule with no total deliberate idle times \( j_0 \) must be executed first and the following holds: \( S(j_0, p(j_0)) = 0 \). As \( p(j_0) \) is a dedicated processor, \( j_k \) cannot be scheduled to \( p(j_0) \). Therefore, \( j_k \) cannot be executed before \( C(j_0, p_0) + d(j_0, j_k) \), with \( C(j_0, p_0) = S(j_0, p_0) + w(j_0) \). If we assume that \( j_0 \) is of zero length and that \( d(j_0, j_k) = r(j_1) \), then \( j_k \) can be started at or after \( r(j_k) \). This proves the equivalence for \( j_k \). This reasoning can be repeated for all the jobs in \( J \).

Note that only a single artificial entry task \( j_0 \) of zero length is both necessary and sufficient for the equivalence to hold for the whole set of jobs with release dates. The outcome schedule on the remaining processors is unaffected by the introduction of the additional processor under the outlined assumptions.

On the basis of the equivalence in Theorem 1, a scheduling problem with release dates may be easily transformed in one without explicit release dates. Therefore, this thesis mainly focuses on algorithms for problems with communication delays but without explicit release dates and the algorithms proposed as contributions do not consider release dates.

2.3.2 Start and Finish Time of Jobs

This subsection provides definitions that are directly applicable as an algorithmic method for the determination of the earliest start time of a job in a precedence-constrained job set and in a generic scheduling problem with and without preemption. The actual start time may not be the earliest possible (it is not, in general, in lazy policies, see 2.16.2). The corresponding finish times can be trivially derived by adding the appropriate job durations.

**Definition 2.14.** The parent set of a job \( j \) is the set \( J_{\text{par}}(j) \) of jobs that are the parents of the job with respect to the set of precedence constraints given by \( E \), that is:

\[
J_{\text{par}}(j) = \{ j_i \in J \mid (j_i, j) \in E \}
\] (2.2)
A job with no parents is also termed an entry node of the task graph. The set of entry nodes is given by:

\[ J_{\text{entry}} = \{ j \in J : J_{\text{par}}(j) = \emptyset \} \]  

(2.3)

Conversely, the child set of \( j \) is the set \( J_{\text{ch}}(j) \in J \) given by:

\[ J_{\text{ch}}(j) = \{ j_i \in J \mid (j, j_i) \in E \} \]  

(2.4)

and the jobs with no children form the set of exit nodes of the task graph.

**Definition 2.15.** Let \( P \) be the set of processors and let \( J_{\text{par}}(j) \in J \) be the parent set of a job \( j \). The earliest feasible start time \( S_{\text{EST}}^{\text{pmtn}}(j,p) \) of \( j \) on a processor \( p \) in a preemptive schedule is given by the maximum of the release date \( r_j \) of \( j \) and the arrival time of the latest communication from all the parents of \( j \):

\[ S_{\text{EST}}^{\text{pmtn}}(j,p) = \max \{ r_j, \max_{j_{\text{par}} \in J_{\text{par}}(j)} \{ C(j_{\text{par}}, p(j_{\text{par}})) + d(j_{\text{par}}, j) \} \} \]  

(2.5)

Omitting release dates as per Theorem 7 (i.e., if \( \forall j \in J \Rightarrow r_j = 0 \)), the earliest feasible start time \( S_{\text{EST}}^{\text{pmtn}}(j,p) \) of \( j \) on \( p \) is given by the latest arrival, to processor \( p \), of any communication required by \( j \) from any of the parents of \( j \):

\[ S_{\text{EST}}^{\text{pmtn}}(j,p) = \max_{j_{\text{par}} \in J_{\text{par}}(j)} \{ C(j_{\text{par}}, p(j_{\text{par}})) + d(j_{\text{par}}, j) \} \]  

(2.6)

where the indexes \( \text{EST, pmtn} \) refer to the earliest feasible start time and preemptive scheduling, respectively. The earliest feasible start time \( S_{\text{EST}}(j,p) \), or availability time, of a job \( j \) on a processor \( p \) in a non-preemptive schedule is given by:

\[ S_{\text{EST}}(j,p) = \max \{ I(p), \max_{j_{\text{par}} \in J_{\text{par}}(j)} \{ C(j_{\text{par}}, p(j_{\text{par}})) + d(j_{\text{par}}, j) \} \} \]  

(2.7)

where \( I(p) \) is the time at which the processor \( p \) first becomes idle, defined by the following equation:

\[ I(p) = \min_j C(j,p) \mid (C(j,p) = S(k,p) \Rightarrow k \in \emptyset) \]  

(2.8)

Note that Eq. 2.8 describes either a closed or a half-open (to infinity) idle time slot on the processor \( p \). As Eq. 2.7 provides a method of finding the earliest feasible start time...
for the job \( j \), it does not require explicitly that the job fit into the idle time slot, if not half-open. Naturally, almost all schedulers working with deterministic activity times do require the idle time slot to be sufficiently large, which can be checked easily, given such deterministic values. In practice, they make additional provisions in this regard, which implies searching for the next large enough slot, if the first one tested is not sufficient. In this thesis, however, the adoption of stochastic activity times (distributions and estimates) motivates a more flexible approach, that is detailed in the following chapters.

The absolute earliest start time \( (AEST) \) of a job is usually defined independently of the scheduling platform (and process), by only considering the precedence constraints and release times. It is relevant for a number of list scheduling heuristics as a numeric attribute taken into account when determining the job priority list. Other such attributes include b-levels, t-levels and others (see Sect. \[2.16.1\]).

**Definition 2.16.** The absolute earliest start time \( S_{AEST}(j) \) of a job \( j \) is given by:

\[
S_{AEST}(j) = \max_{j_{par} \in J_{par}(j)} \{ S_{AEST}(j_{par}) + w(j_{par}) \} \tag{2.9}
\]

The following equivalence holds.

**Theorem 2.** The absolute earliest start time \( S_{AEST}(j) \) of a job \( j \) is alternatively given by the earliest feasible start time of a job \( j \) on any of an unbounded number of processors, in a preemptive schedule, with communication times set to zero:

\[
S_{AEST}(j) = \min_{p \in P} S_{pmtn}^{EST}(j, p) \mid (|P| = \infty \land \forall k, i : d(k, i) = 0) \tag{2.10}
\]

**Proof.** The proof follows directly from Definitions \[2.7\] and \[2.9\].

Note that the same algorithmic methods can be carried over to stochastic settings, which implies performing the same operations on the corresponding stochastic variables. This is an exact method, which, however, is computationally very expensive (\[96, 97\], \[2.15.1\]). For this reason usually appropriate estimates in the form of scalars (i.e., single real-valued numbers) are chosen to approximate and replace the (distributions of the) stochastic variables of interest. The very same methods for the computation of the (estimated) job start and finish times are applicable when using such estimates.
2.4 Scheduling Policies

Let $P$ be an instance of a scheduling problem. In this thesis, the following broad definition of policy is used.

**Definition 2.17.** A (scheduling) policy $\pi$ is any algorithmic method that derives a unique solution $S$ for a scheduling problem $P$, as per Definition 2.11.

It follows from Definitions 2.11, 2.17 that $\pi$ entails a universally measurable function $S_i(\pi)$, which maps the realizations $w(j)$ of the processing times of all jobs $j$ to the corresponding start times of the jobs (hence the use of $S_i(\pi)$ instead of just $\pi$, as the allocation to the processors is not relevant for the discussion at the moment). Formally, $\mathbb{R}_+^n \rightarrow \mathbb{R}_+^n$, $w(j) \xrightarrow{S_i(\pi)} S_j$, where $\mathbb{R}_+ = \{v \in \mathbb{R} : v \geq 0\}$.

The requirement that $S(\pi)$ be universally measurable guarantees the existence of the distribution of the objective function value associated with $\pi$ and needs to be stated explicitly, as it does not follow from the definitions of schedule and policy (proof by example provided by Kaerkes et al. [123], following [236]). Naturally, policies required to generate complete, feasible schedules must respect additional constraints as per Def. 2.12.

For an alternative definition based on the view of a scheduling problem as a sequential decision problem, see Sect. 2.11.

A policy may be **offline**, in which case the term *algorithm* is more frequently used in literature, or **online**. The distinction between offline or online scheduling problems is, intuitively, related to the relationship of the process of scheduling to the execution of the jobs $J$ in $P$.

**Definition 2.18.** A policy $\pi$ is offline if it computes the solution $S$ to a scheduling problem $P$ before the execution of any job in $J(P)$ begins, or not taking this execution into account.

A policy $\pi$ is online if it computes $S$ during the execution of the job set $J(P)$ and is defined for a set of moments in time. For each such moment an online policy performs an action.

Offline algorithms originated in deterministic scheduling problems and are applied mostly when determinism of both the processing and communication times can be
guaranteed. On the other hand, many prominent authors from the field of stochastic scheduling advocate the view that the passage from deterministic to stochastic problem settings, and the stochastic environments in general, require the exclusive use of online policies. To accommodate this view, they also adopt a narrower definition of policy, as in Uetz [236] and Radermacher [196], adding the requirement that the policy be non-anticipative (see Definition 2.44 in Sect. 2.11).

Online policies usually take advantage of the information about the set of jobs and their specific characteristics that becomes available at execution time and, commonly, provide a rule to schedule the next available job one at a time, i.e., they compute the solution step by step. Sect. 2.13 provides more details about the above distinction by referencing different offline and online scheduling models: the models referred to there not only represent features of possible policies, or requirements for policies, but provide a classification scheme for the amount of information available for a policy to work with.

Another important distinction of particular interest for online policies is the one between elementary and non-elementary policies, formalized by Möhring, Radermacher and Weiss [171].

**Definition 2.19.** Let \( \{j_i\} = J \) be a set of jobs and let \( S_{EST}(j) = \min_{p \in P} S_{EST}(j, p) \) be the time when job \( j \) first becomes available on any processor. A policy is elementary if it is defined only for the set \( t \in \{ \forall j \in J : t = S_{EST}(j) \} \), non-elementary otherwise.

It follows that elementary policies only make decisions for states associated with specific changes in the state of the world that result in a new job becoming available, such as the release of a new job or the completion of a job under processing. In other words, they operate on a finite, discrete event set. In contrast, non-elementary policies can be defined for any moment in time. Non-elementary policies, in general, make sense when the passage of time alone has the potential of providing meaningful information to guide the scheduling decisions (see, e.g., the examples in Uetz [236]). On the other hand, from the above definition it is obvious that there exists an infinite number of such policies, which may make their formulation complex (see also Sect. 2.11 on this regard), unless a really trivial mechanism is used (e.g., a fixed-period clock). In practice, non-elementary policies are seldom studied in the literature. On the other hand, this thesis reports an improved online scheduling policy based on the idea of taking into
consideration the elapsed time of the activities still running at the time a new decision is made: the computations, however, are performed in a fashion specific for elementary policies (see Chapter 4).

Möhring, Radermacher and Weiss [172] additionally define the so called set policies (see definition below), which are guaranteed to be optimal for regular additive objective functions under specific assumptions (see Sect. 2.6).

**Definition 2.20.** A set policy is an elementary policy whose action at any time \( t \) may only depend on the set of jobs completed by \( t \), and the set of jobs that are still in process at \( t \), but not on the realizations \( r(w_j) \) of the stochastic processing times \( w_j \) or the time \( t \) itself.

### 2.5 Schedule Performance and Optimality

In this section, the notions of optimality of a schedule and of a policy are briefly presented, for the single- and multi-objective case. In the following, it is assumed that the optimization in question equates to minimization: A formulation for a maximization problem is analogous.

As a “schedule” is a solution to a scheduling (optimization) problem, in this section, the terms “schedule” and “solution” are used interchangeably and both are indicated by the symbol \( S \). Moreover, “optimal”, “optimization” and related terms always implies “minimal”, “minimization” and the corresponding terms, unless explicitly specified otherwise.

**Definition 2.21.** Let \( P \) be a scheduling problem instance with an objective function \( \gamma \). A schedule \( S^* \) is optimal with respect to \( \gamma \), if it satisfies:

\[
S^*(P) = \inf_{S \in \{S(P)\}} \gamma(S(P)) \quad (2.11)
\]

where \( \{S(P)\} \) is the set of all solutions to \( P \).

**Definition 2.22.** It clearly follows from of the above that a scheduling policy \( \pi^* \) is optimal with respect to \( \gamma \), if it satisfies:

\[
\pi^* = \inf_{\pi \in \Pi} \gamma(S(\pi, P)) \quad (2.12)
\]
where \( S(\pi, P) \) is a solution to \( P \) obtained using a policy \( \pi \).

### 2.6 Relevant Functions in Single-Objective Scheduling

In this section, some of the canonical objective functions of interest in the area of scheduling for parallel processing and of particular relevance for this thesis are preliminarily presented. The next few sections introduce examples and important results based on them, while Sect. 2.12 provides a more systematic view.

**Definition 2.23.** The sum of (weighted) completion times is defined as follows, using notation in Def. 2.12:

\[
\sum_j w_j C_j \quad (2.13)
\]

The unweighted sum of completion times is also termed flow time. Note that \( w_j \) is assumed not to be a function of \( C_j \), thus both sums are linear functions of \( \{C_j\} \). By definition, this function is additive; it is also non-decreasing, which in literature is sometimes referred to as “regular” (e.g., in Uetz [237]). Möhring, Schulz and Uetz [172] prove that, under the assumption of exponential processing times, online scheduling problems with objective functions defined as expectations on regular additive objectives like the weighted sum, are known to have an optimal policy within the class of set policies (see Def. 2.20). Uetz [237] observes that such a policy must generally deliberately leave some processors idle at times, i.e., it is a deliberate-idle time or lazy policy (see 2.16.2).

**Definition 2.24.** The overall schedule length (or makespan) \( C_{\text{max}} \) is the total time required for an application to complete. It is given by:

\[
C_{\text{max}} = \max_j \{C_j\} \quad (2.14)
\]

Note that the makespan \( C_{\text{max}} \) is a non-linear function of \( \{C_j\} \) (in Uetz’s nomenclature, it is neither additive nor regular). Another notable fact is that optional communication delays make also the weighted sum of completion times non-additive with respect to
Chapter 2. Background and Preliminaries

the completion times and therefore not subject to the guarantee formulated by Möhring, Radermacher and Weiss \[172\].

When examined in the context of a queue of workflow instances, i.e., from the point of view of queuing theory (see 2.18), which abstracts the single activities within the workflow away, the (average) makespan may be viewed as the (average) service time of a workflow instance (see Chapter 4).

**Definition 2.25.** The speed-up of an application when executed on a given platform $L$ with respect to another platform $L'$ is the ratio of the total processing time (schedule length) on $L'$ and $L$:

$$
\Delta C_{\text{max}}(L', L) = \frac{C_{\text{max}}(L')}{C_{\text{max}}(L)} \quad (2.15)
$$

### 2.7 Multi-Objective Scheduling

This section introduces some of the notions and techniques employed in multi-objective scheduling. Some notions, such as those related to the performance space of a policy, are more naturally defined in the multi-objective context, but carry over well to single-objective scheduling problems in stochastic settings. Moreover, techniques such as the employment of parameter-driven aggregation functions as a method of solving a multi-objective problem by reducing it to a single-objective one are also fundamental for the work presented in this thesis.

Willborn and Cheng \[262\] observe that, in general, quality (and therefore, quality of service, or QoS) is a multidimensional concept. Therefore, the set of objective functions to be simultaneously optimized in a scheduling problem can be defined with respect to multiple resources as defined in Sect. 2.2, e.g., (weighted) number of processors used (i.e., pertaining to renewable resources) or weighted or maximum job completion time, quality or any other generic QoS attribute (all of which can be modeled as non-renewable resources).

Even in “classical” (completion time-oriented) scheduling for parallel processing the number of definable scheduling problems is virtually infinite, as has already been indicated by Lawler et al. \[145\]. This is naturally more apparent, when criteria related
to multiple, heterogeneous resources are considered. For an extensive discussion and survey of the state of the art in multi-objective (or multi-criteria) scheduling, the works of, e.g., T’kindt [231] and Hoogeveen [105] should be consulted.

It should be clear, keeping the complexity of seemingly easy single-objective scheduling problems in mind, that the naive extension of Definition 2.21 to the multi-objective case, given in Definition 2.26, must create substantial problems.

**Definition 2.26.** A solution $S^*$ is optimal with respect to a set of objective functions $\Gamma$, if and only if:

$$\forall \gamma \in \Gamma \Rightarrow \gamma(S^*) = \inf_{S \in \{S(P)\}} \gamma(S).$$

(2.16)

A solution in the sense of the above definition seldom exists in practice: intuitively, most realistic problems have conflicting objectives. More generally, except in trivial cases, the optimality, or any performance level, of a solution with respect to one objective function does not imply the optimality or any performance guarantee with respect to any other. In the same way, there is, in general, no relationship between the performance space (defined below) of a policy, given a class of scheduling problems, with respect to an objective, with the likewise defined performance spaces for any other objective.

**Definition 2.27.** The performance space of a policy $\pi$ is the set of values of an objective function $\gamma$ for the solutions $\{S(P, \pi)\}$ given by $\pi$ to a set of problems $\{P\}$.

In some problems, especially when the set of objective functions is relatively small, there exist solutions that are dominated by others.

**Definition 2.28.** A solution $S'$ is dominated by a solution $S$, written $S < S'$, with respect to a set of objective functions $\Gamma$, if and only if:

$$\forall \gamma \in \Gamma \Rightarrow \gamma(S) \leq \gamma(S') \land \exists \gamma^* \in \Gamma : \gamma^*(S) < \gamma^*(S').$$

(2.17)

Where dominated solutions exist, these can be eliminated from the solution set. To individuate the actual solution(s) of interest, two basic techniques are used:

\[^{2}\text{In the following, as in the previous section, “optimum” is assumed to be a synonym of “minimum”, with all the obvious implications.}\]
Chapter 2. Background and Preliminaries

- a full or partial enumeration of the solutions, with the rationale that, in the absence of further information, the choice of the final solution is external to the optimization problem
- an automated choice of a solution that is optimal with respect to a modified objective function with or without additional constraints.

The first approach is, however, usually impracticable for very large solution sets, especially in many-objective problems that tend to lack dominated solutions to discard (see, e.g., Corne and Knowles [57], Ishibuchi and Tsukamoto [115] or Sato, Aguirre and Tanaka [212]). It is also of no practical importance if the optimization problem at hand requires the possible solutions to be evaluated as part of a sequential decision problem, such as a scheduling problem (see 2.11): at each decision step only one solution must be chosen and this choice needs to be made algorithmically and promptly.

In the second approach, the objective function, as outlined above, is replaced by a modified one, which is an ordering or an aggregation of the objective functions in \( \Gamma \), with the optional introduction of a set of constraints. The point is to make dominance is always decidable. Typically the following techniques are used:

- a lexicographic order of the objective functions, i.e., given a total order \( \forall j > i \Rightarrow (\gamma_i, \gamma_j, \prec) \), it is the case that: \( S < S' \Leftrightarrow [\forall i \Rightarrow \gamma_i(S) < \gamma_i(S') \lor (\gamma_i(S) = \gamma_i(S') \land \exists j > i : \gamma_j(S) < \gamma_j(S'))] \)

- an aggregation function (typically linear), i.e., given a set \( \{\alpha_i\} : \forall i \alpha_i \in \mathbb{R} \) and \( f(S) = \sum \alpha_i \gamma_i(S) \), it is the case that: \( S < S' \Leftrightarrow f(S) < f(S') \)

- \( \epsilon \)-constraint, i.e., given \( \gamma_i \in \Gamma \) for some \( i \) and given a set \( \{\epsilon_j\} : \forall j \Rightarrow \epsilon_j \in \mathbb{R} \), it is the case that: \( S < S' \Leftrightarrow [\gamma_i(S) < \gamma_i(S') \land \forall j \neq i \Rightarrow \gamma_j(S) \leq \epsilon_j] \).

### 2.8 Notable Issues in Bi-Objective Scheduling

As it is fairly obvious, given the arguments in the previous section, the set of methods applicable to all or most multi-objective problems must be very restricted. In fact, in the broad field of multi-objective optimization there are attempts at formulating general-purpose methods of deriving the Pareto set or its approximation (see, e.g.,
2.8. Notable Issues in Bi-Objective Scheduling

Cheng, Janiak and Kovalyov [50]. The hypervolume indicator is quite established as a metric of the goodness (fitness) of the solution (see, e.g., Auger et al. [19], Zitzler, Brockhoff and Thiele [278] and others). In multi-objective scheduling specifically, considerable efforts have been devoted in recent years to the problem of approximability (see, e.g., Papadimitriou and Yannakakis [183]) and defining approximation classes for multi-criteria scheduling problems (see, e.g., Angel, Bampis and Kononov [13]). All of these approaches are motivated by the necessity of working around the lack of methods of solving optimization problems that would be at the same time generic (effective for entire classes of problems), provably near-optimal and efficient.

For this reason, in scheduling, much effort has been dedicated to particular bi-criterion problems, whose specific structure could be exploited in the solution. Initially priority was given to problem settings that are more amenable to analysis, such as single-machine problems, see, e.g., Hoogeveen [106] and Lee and Vairaktarakis [149]. Cheng, Janiak and Kovalyov [50] presented various problems formulated with respect to two criteria, one of which is relative to a renewable resource, in their case, the maximum number of processors utilized, and one is a “regular scheduling criterion”, i.e., it is one of the classical functions of the (non-renewable) processing time of the jobs. An example was the problem of scheduling a set of precedence-constrained jobs such that a variable number of processing resources could be assigned to each job, resulting in a linear speed-up given by a known factor. The objective was to minimize the makespan and the maximum number of resources utilized. The authors also provided a general method of deriving the Pareto set or its approximation. Note that in these example approaches the optimization criteria pertain to different resources.

In scheduling, arguably unlike in other optimization problems, it is also possible, and often the case, that multiple objectives are defined with respect to the same resource, e.g., schedule length and flow time (the sum of the jobs’ completion times). In a manner that is perhaps contrary to the intuition, a simultaneous minimization of both objectives is generally not possible. In fact, these two objectives correspond to different aggregation functions of the jobs’ finish times, i.e., different functions defined in the time domain (see Sect. 2.6 for their properties, such as regularity and related concepts).

In practice, as McCormick and Pinedo [167] proved by example, the solution $S_{\sum C_j}$ to the problem $\sum |\text{prec}| \sum w_j C_j$ is not a solution to the corresponding problem $|\text{prec}| C_{\text{max}}$ or $|\text{prec}| \text{avg}[C_j]$ for a given set of weights $\{w_j\}$. The converse is also true. This is
due to the combinatorial nature of the problem of scheduling precedence-constrained job sets, which makes the bi-criterion problem hard even on a single machine and without communication delays.

Stein and Wein [225] first formulated $\sum_{\text{prec}} C_{\text{max}}, \text{avg}[C_j]$ as a bi-objective problem and provided results about its approximability. Improved results are due to Aslam et al. [16]. Hurkens and Coster [111] investigated the same problem with the added complexity of a setting with (multiple parallel) unrelated machines. More recently, Bunde [42] investigated the same problem in the grid context, where power consumption is additionally an issue and thus power awareness criteria are added. Many seemingly easy problems, even among those related to simple linear aggregation functions of the jobs’ completion times, are in fact hard. For example, Angel, Bampis and Gourves [12] formulated the average completion time problem constrained by a bound on the flow time, i.e., $\sum_{\text{prec}} C_i$ subject to $\sum C_i < B$, where $B \in \mathbb{R}_+$ is a fixed bound.

For a Pareto-inspired approach that aims at a broader class of problems with simultaneously optimized criteria that are all related to processing time, see Bilò, Flammini and Moscardelli [31]. Finally, for a broader discussion of related multi-objective problems see Dutot et al. [64].

A similarly counter-intuitive case is related to the expected response time of the system (the objective function optimized in Chapters 4 and beyond), which is the sum of the expected execution time and in-queue waiting time (see 2.18). The intuition behind the proof is the following. Let us assume that at the moment when a scheduling decision is made there are two workflow instances in the system, one ($I_1$) already executing and a second one ($I_2$), waiting, and both have at least one job ready for execution on some processor, which is idle. A first scheduling policy $\pi_1$ strives to minimize $C_{\text{max}}(I_1)$, aiming at completing the instance already running as soon as possible. Clearly, it will choose to execute the next job of $I_1$ first. Another policy $\pi_2$ minimizes the waiting time of the second instance ($W_q(I_2)$). It will opt for the execution of the first job of the second instance as early as possible. The second policy, while admittedly myopic, will (indiscriminately) minimize the waiting time (not just for $I_2$, but on average). The average waiting time, intuitively, will be close to the average execution time of the first job of the workflow. Clearly, $\pi_2$ may cause the makespan of an instance to grow indefinitely. Conversely, $\pi_1$ will keep the makespan constant, while the waiting time may increase across instances, if the arrival rate is too high. This illustrates that the
two objectives whose sum is the total response time of the system may be conflicting. In other words, minimizing the expected response is in fact a bi-objective optimization problem. However, the two objectives may be jointly expressed by a linear aggregation (see [2.7]), for the purpose of finding a solution, and thus transformed into a single objective.

Apart from the expected response time problem briefly introduced here, this thesis proposes to treat the (global) expected makespan optimization problem in separation (more accurately, a reformulation of this problem) as a bi-objective problem in its own right. Namely, the global makespan minimization problem is mapped to a problem of locally (at each decision step) minimizing two objective functions: the earliest finish time of the task considered for scheduling at the given step and the delay that may be caused by this decision in the finish times of any higher priority task. The relative importance of the two objectives changes throughout the scheduling process and, obviously, reflects the relative importance of the considered tasks. Naturally, it is a heuristic approach. However, it is well-fitted within the framework of list scheduling and, though without performance guarantees, is particularly suited to handle stochastic activity durations. For details, see Chapter 3.

2.9 Probability-Theoretical Background

This section briefly summarizes the notions of probability theory that are related to the description of resource utilization associated with the activities in all application models used throughout this thesis.

These notions and theorems build the basis for the results in Sect. 2.15.1 that pertain to the problem of computing or approximating the distribution of the overall schedule length, when the activity times are distributed stochastically. Notably, the same problem resurfaces each time during the scheduling process when a distributed partial schedule length is used in the computation of a job’s priority. To solve the latter problem efficiently, the full probability distributions of the stochastic variables of interest are commonly replaced with estimates, which are appropriately computed scalar values ([155], [160], [227], [153]). This section also provides the necessary background for the derivation of such estimates.
The following definitions use the classical probability theory formulation. For the purposes of this thesis, it is sufficient and slightly more straightforward than the more recent measure-theoretic definition set, which, in this context, would be equivalent.

**Definition 2.29.** The cumulative density function (CDF) of a random variable X is a function $F(x)$ such that:

$$F(x) = \Pr[X \leq x]$$

(2.18)

If the CDF is absolutely continuous, the random variable in question is continuous, and also admits a probability density function.

**Definition 2.30.** The probability density function (PDF) of a random variable X is a non-negative Lebesgue-integrable function $f(x)$ such that:

$$\Pr[a \leq X \leq b] = \int_a^b f(x)\,dx$$

(2.19)

If the PDF is continuous at $x$, then:

$$f(x) = \frac{d}{dx} F(x)$$

(2.20)

Conversely:

$$F(x) = \int_{-\infty}^t f(t)\,dt$$

(2.21)

In some cases, the calculations are made easier by the adoption of the moment generating function (MGF), defined below (this alternative formalism is continued in later sections).

**Definition 2.31.** The moment generating function (MGF) of a random variable X is the expectation (if such an expectation exists) of the value of the simple exponential function of the product $tX$, where $t \in \mathbb{R}$ is a free parameter. Thus the MGF is a function $M(t)$ such that:

$$M(t) = E[e^{tX}]$$

(2.22)

**Definition 2.32.** If a PDF exists, then the mean, or expected value, or first moment of the random variable, indicated by $E[X]$ or $\mu$, if it exists, is given by:

$$E[X] = \int_{-\infty}^\infty x f(x)\,dx = \int_{-\infty}^\infty x dF(x)$$

(2.23)
2.9. Probability-Theoretical Background

**Definition 2.33.** The \( n \)-th moment about the mean, or \( n \)-th central moment, of a random variable \( X \) with a PDF given by \( f(x) \), is:

\[
\int_{-\infty}^{\infty} (x - \mu)^n f(x) \, dx
\]  

(2.24)

In particular, the second central moment about the mean, or variance of \( X \), indicated by \( \text{Var}[X] \) or \( \sigma^2 \), is:

\[
\text{Var}[X] = \sigma^2 = \int_{-\infty}^{\infty} (x - \mu)^2 f(x) \, dx = E[(X - \mu)^2] = E[X^2] - \mu^2
\]  

(2.25)

**Definition 2.34.** The standard deviation \( \sigma \) of a random variable \( X \) is the square root of its variance:

\[
\sigma = \sqrt{\text{Var}[X]}
\]  

(2.26)

The standard deviation is a measure of the average dispersion of the random variable with respect to its mean, which is universally used in probability calculus and applications, including, e.g., as a measure of robustness in scheduling solutions (see Sect. 2.17). However, it is important to note that this measure is non-directional, i.e., does not contain the information on the portion of the dispersion interval lying above (or below) the mean.

An important property, valid for any stochastic variable with known mean and standard deviation, is stated by the Chebyshev inequality.

**Chebyshev’s inequality.** The probability that the value of a stochastic variable \( X \) lies no closer to its mean than \( k \) standard deviations, with \( k \in \mathbb{R}_+ \), is not greater than the inverse square of \( k \):

\[
\Pr[|X - \mu| \geq k\sigma] \leq \frac{1}{k^2}
\]  

(2.27)

This simple inequality already provides means of approximating the result of the process of drawing samples from any distribution, when it is not given in a closed form, whether such a form is not known or known not to exist. The approximation may be poor, which naturally depends on the factor \( k \).
2.9.1 Relevant Properties of Independent Random Variables

The computation or evaluation of the length of a stochastic path in a task graph is, in general, a \#P-complete problem ([96, 97]), i.e., strictly harder than any problem in \textit{NP}. In many approaches found in the literature (see Sect. 2.15.1), results pertaining to such evaluation (exact or approximate) are obtained under the simplifying assumption of independence of the stochastic variables of interest. On the one hand, this assumption does not always make the problem much easier, on the other there are many cases in which it does in fact not hold. These two correlated issues are also discussed in more detail in the above referenced section. In many simpler cases, however, independence can both be guaranteed at least for the stochastic activity durations considered in separation, i.e., abstracting away the task graph, and is helpful in the evaluation of some stochastic paths. Therefore it is worth summarizing here the relevant probability-theoretical background.

Definition 2.35. Two random variables \( X, Y \) are independent if and only if, for all \( a, b \in \mathbb{R} \), the events \( A = \{ X \leq a \} \) and \( B = \{ Y \leq b \} \) are independent, i.e., if and only if:

\[
Pr(A \cap B) = Pr(A)Pr(B)
\]  

(2.28)

The following results are theorems valid for independent random variables (IRVs): for proofs, see, e.g., Boes, Graybill and Mood [36].

Sum of IRVs. If two continuous random variables \( X, Y \) are independent and both admit a PDF, then, given the respective PDFs \( f_X(x) \) for \( X \), \( f_Y(x) \) for \( Y \) and \( Z = X + Y \), \( Z \) is a continuous random variable that has a PDF given by:

\[
f_Z(x) = \int_{-\infty}^{\infty} f_X(t)f_Y(x-t)dt = f_X(x) \ast f_Y(x)
\]  

(2.29)

The “\(*\)” operator in the above equation, with the meaning defined on the right hand side, is termed convolution. In practical implementations the computation signified by it is performed using a fast Fourier transform (FFT) applied to a discrete representation of the relevant PDFs (see Sect. 2.15.1).

Maximum and minimum of IRVs. If two random variables \( X, Y \) are independent, then, given the respective CDFs \( F_X(x) \) for \( X \), \( F_Y(x) \) for \( Y \) and given \( Z = \)
$\max\{X,Y\}, Z' = \min\{X,Y\}$. $Z$ and $Z'$ are two random variables having a CDF $F_Z(x), F_{Z'}(x)$ given by:

\begin{align}
F_Z(x) &= F_X(x)F_Y(x) \\
F_{Z'}(x) &= [1 - F_X(x)][1 - F_Y(x)]
\end{align}

For the purposes of the presentation of the theoretical foundations of the approaches presented in this thesis, theorems valid for absolutely continuous univariate distributions are mostly used. Multivariate distributions are not employed. The above theorems and derived ones, however, are valid for discrete random variables, when formulated using the discrete analogues of the PDF (i.e., the probability mass function, PMF) and of integration (i.e., the sum).

### 2.10 Performance Space and Optimality in Scheduling

**Definition 2.36.** Let $P$ be a scheduling problem, $\pi$ be a policy, $\rho$ a resource and $\gamma_\rho(\pi)$ an objective function defined in such a way that its distribution with respect to $\pi$ exists. Let $w_\rho(a)$ be the random variable that describes the resource requirement value for an activity $a$ and $w_\rho$ the vector of such resource requirement variables for all the activities in $P$. Let $r(w)_\rho$ be a realization of $w_\rho$ (i.e., a vector containing a realization for each single variable $w_\rho(a)$). The performance space of a policy $\pi$ with respect to a set of inputs $\{r(w)_\rho\}$ of cardinality $R$ is given by the set:

\[ \Gamma_\rho(\pi, r(w)_\rho) = \{\gamma_\rho(\pi) | w_\rho = r_k(w)_\rho, \forall k \leq R\} \]  

with $|\Gamma_\rho(\pi, r(w)_\rho)| = R$.

Naturally, in most realistic problems, the elements of both $\{r(w)_\rho\}$ and $\Gamma_\rho(\pi, r(w)_\rho)$ assume non-negative real values. For a concrete example of what $\rho$ and the corresponding vectors may represent, let us consider the following. For the makespan objective, $\rho$ is the processing time and $\gamma_\rho = C_{\max}$. The performance space of a policy with respect to the objective is given by the set of values that the makespan assumes under the policy, when its input is a set of realizations of the processing times of the activities. Throughout this thesis, it is assumed that any realizations are drawn from (are con-
sistent with) probability distributions known \textit{a priori}, but approaches that avoid this assumption are possible, when no such information is accessible (see Sect. 2.13).

Analogously to the case of the multi-objective problem (Def. 2.26), the following optimality definition could be put forward (as usual, “optimum” implies “minimum” in the following).

\textbf{Definition 2.37.} A policy $\pi^\ast$ is performance-space optimal with respect to an objective function $\gamma_\rho$, if and only if:

$$\forall r(w)_\rho \Rightarrow \gamma_\rho(\pi^\ast, r(w)_\rho) = \inf_{\pi \in \Pi} \gamma_\rho(\pi, r(w)_\rho). \quad (2.33)$$

Naturally and much like in the multi-objective context, such policies may exist only for the most trivial problems.

The following definition, put forward by Pinedo [189], is milder, but still yields relatively few practical results, tractability issues apart, as noted by the author.

\textbf{Definition 2.38.} A policy $\pi^\ast$ is probabilistically optimal with respect to an objective function $\gamma_\rho$, if and only if, for any non-negative real number $x$, given an event in which the objective value $\gamma_\rho(\pi, w_\rho)$ exceeds $x$, the probability of such an event is the smallest for $\pi^\ast$:

$$\forall x \geq 0 \Rightarrow \Pr[\gamma_\rho(\pi^\ast, w_\rho) > x] = \inf_{\pi \in \Pi} \Pr[\gamma_\rho(\pi, w_\rho) > x]. \quad (2.34)$$

As optimality in a stronger sense cannot be achieved, the research is concentrated on finding policies whose performance spaces have desirable properties, in particular such that a certain function defined on the space attains its minimum. In most of the literature, the expected value of the objective function (actually, of the distribution of its values) is assumed to be a representative measure of the performance (in the common sense of the word) and the standard deviation of the distribution is viewed as a meaningful indication of the robustness of a policy (see Sect. 2.17).

The following provides a definition of a weaker, expectation-based notion of optimality.

\textbf{Definition 2.39.} A policy $\pi^\ast$ is optimal in expectation with respect to an objective function $\gamma_\rho$, if and only if:

$$\forall r(w)_\rho \Rightarrow E[\gamma_\rho(\pi^\ast, w_\rho)] = \inf_{\pi \in \Pi} E[\gamma_\rho(\pi, w_\rho)]. \quad (2.35)$$
According to a result derived by Möhring, Radermacher and Weiss in [171], a policy optimal in the sense of Def. 2.39 always exists, if the objective function is lower semicontinuous or if the probability distribution of $w_\rho$ is finite discrete or has a Lebesgue density. This is the case for the objective functions considered in this thesis. Notably, the policies examined throughout the thesis are all aimed at optimizing objective functions defined in terms of expected values. However, the measures of the corresponding standard deviations, as a robustness metric, are additionally provided (see Sect. 2.17). Moreover, the performance spaces of the policies are characterized in more detail by means of five-number summaries.

2.11 Scheduling as a Sequential Decision Process

Once the notions of schedule, policy and optimality are defined, it is possible to provide an alternative formulation of the schedule optimization problem as a sequential decision process and, specifically, as a Markov decision problem (MDP). This formulation gives valuable insights for the understanding of the complexity of the optimization problem and of the issues related, e.g., to the existence of non-elementary policies (see 2.19). It is also at the root of some approaches to the scheduling problem, such as stochastic scheduling and dynamic programming, as well as other approaches, where the MDP model is applied directly to problems with well-behaved objective functions.

Definition 2.40. Given a discrete set of time indexes $0, \ldots, t, t + 1, \ldots \in \mathbb{N}$, a Markov decision process (MDP) is a tuple $< s_0, T(s, a, s'), R(s) >$, where $s_0$ is the initial state, $s$ is the state at some moment $t$, $s'$ is the state at the moment that immediately follows $t + 1$, $T(s, a, s') = Pr[s_{t+1} = s'|s_t = s, a_t = a]$ is the transition model from a state $s$ to $s'$, given an action $a$, and $R(s) = R(s, s') = R_a(s)$ is the immediate reward expected after the transition from $s$ to $s'$.

Naturally, an MDP represents a class of decision problems (multi-stage decision problems) that is much broader than that of scheduling (decision) problems. In fact, an MDP (optimization) problem in general is formulated in terms of maximizing the expected sum of rewards over a finite or infinite horizon (see definition below).

A policy in the MDP sense is a solution that specifies a decision for every state that may be reached (Russell and Norvig [205]). Formally, the following definition holds.
Definition 2.41. A policy $\pi$ in an MDP is a method of choosing an action $a$ in a state $s$, i.e., a function $\pi(s_t) = a$.

Definition 2.42. The utility of a sequence of states in an MDP over a horizon $h \in \mathbb{N} \cup \{\infty\}$ is given by:

$$U_h = \sum_{t=0}^{h} \kappa^t R(s_t)$$

where $\kappa \in \mathbb{R}$ with $0 < \kappa \leq 1$ is the discount factor. The utility clearly depends on the chosen policy. Policies defined for an infinite horizon are called stationary. Conversely, those defined for finite horizons are termed non-stationary.

Definition 2.43. An optimal policy $\pi^*$ is a policy that maximizes the expected utility over the given horizon:

$$\pi^* = \sup_{\pi} E[U_h|\pi] = \sup_{\pi} E[\sum_{t=0}^{h} \kappa^t R(s_t)|\pi]$$

An alternative formulation requires the optimal policy $\pi^*$ to minimize the expected cost over the given horizon:

$$\pi^* = \inf_{\pi} E[G_h|\pi] = \inf_{\pi} E[\sum_{t=0}^{h} \kappa^t g(s_t)|\pi]$$

where $g(s_t) = -R(s_t)$ is the cost of being in the state $s_t$.

The analogy of the optimization problem in the above definition with stochastic scheduling with a flow time objective (sum of completion times) or sum of weighted (discounted) completion times is obvious. In fact, in such a problem, we have:

$$\pi^* = \inf_{\pi} E[\sum_{j=0}^{|J|} w_j C_j|\pi]$$

If the set of job indexes is chosen in a manner that they can correspond to the set of time indexes in the MDP, the problem of minimizing expected sum of weighted completion times of the jobs can be formulated as an MDP with cost minimization (Eq. 2.39). Clearly, the Markov property must also hold: the next state $s'$ may only depend on the current state $s$ and action $a$, not on the past states. This, however, can be guaranteed.
by an appropriate choice of the state model: if the next state depends on past states or actions, those can be included in the current state \[150\], although at the expense of complexity added to the representation of the state. Clearly, the feasibility conditions must be captured in the state transition model.

In his thesis, Uetz \[236\] proposes such a model, embedding an (online) stochastic scheduling problem with an objective formulated over expectations on a regular, additive criterion (objective function), into the stochastic dynamic optimization framework. In this model, the state of the system at a time \( t_i \) consists of the set of all the jobs started or completed by \( t_i \), with their start times and completion times. The action at \( t_i \) is comprised of the set \( B(t_i) \) of jobs scheduled at \( t_i \) and of a future decision time \( t_{i+1} > t_i \). \( B(t_i) \) is derived by the scheduling policy: it must be a function of the feasibility constraints and it may be empty. In particular, a lazy policy may decide not to schedule any jobs, even if some of them are available.

The value of \( t_i \) prescribes the latest moment at which the next action is to be performed. It may be infinite: this indicates an elementary policy, which is one that waits for the occurrence of the next relevant event. Such an event coincides with the potential release of a job, either through a release date or the completion of a parent job. Notably, given the above definition of the state, in Uetz’s stochastic scheduling approach the policy determining \( B(t_i) \) is required to be non-anticipative, i.e., to satisfy the following definition.

**Definition 2.44.** A policy \( \pi \) is non-anticipative if its action at any time \( t \) depends only on the set of jobs completed by \( t \), and the set of jobs that is still in process at \( t \), but not on the realizations of processing times or the time \( t \) itself.

There are some issues with the stochastic scheduling formulation outlined here. Firstly, the analogy between the MDP minimization problem formulation and the scheduling problem with a sum of weighted job completion times does not carry over when the objective function is changed, for instance, to makespan minimization, as the makespan is non-additive with respect to the job completion times. Secondly, an MDP formulation by itself is not necessarily helpful in solving the problem. Thirdly, non-elementary policies (see \[2.4\]) are difficult to capture with the MDP formulation: a limit has to be imposed on the number of states considered. Finally, the MDP formulation explicitly requires the scheduling problem to be one of optimization (specifically, of maximiza-
Despite these difficulties, the MDP formulation is at the root of two prominent approaches to scheduling (and optimal process control in general): stochastic programming and dynamic programming, see Lee and Lee [150]. In multi-stage stochastic optimal control, the main means of reducing the complexity is to limit the number of stages (or, equivalently, the size of the horizon): see, e.g., Infanger [114], for a linear programming approach exploiting this idea. In dynamic programming, the idea is to exploit the celebrated Bellman equation [26], which can be done based on the formulation (below) of the utility of states (see, e.g., Bertsekas [29]).

**Definition 2.45.** The utility $U(s)$ of a state $s$ is given by the sum of the immediate reward $R(s)$ for being in the state $s$ and of the expected discounted utility of the next state $s'$, assuming that the optimal action (of all possible values of $a$) is chosen:

$$U(s) = R(s) + \gamma E[\max_a \sum_{s'} T(s, a, s') U(s')]$$

(2.40)

Equivalently, in cost minimization terms, the total cost $G(s)$ for the current state $s$ is given by the sum of the cost $g(s)$ of being in the state $s$ and the discounted expected cost-to-go, assuming that the optimal control (action) is chosen:

$$G(s) = g(s) + \gamma E[\min_a \sum_{s'} T(s, a, s') G(s')]$$

(2.41)

The optimal policy $\pi^*$ is one that, in the state $s$, for the immediately following state $s'$, satisfies:

$$\pi^* = \sup_{\pi} E[\sum_{s'} T(s, a, s') U(s')] = \inf_{\pi} E[\sum_{s'} T(s, a, s') G(s')]$$

(2.42)

Solving the Bellman equation (in any formulation) for all states is equivalent to solving the corresponding optimization problem. This is usually achieved by means of any of the classical dynamic programming (DP) numerical approaches, such as value or policy iteration, or by a variant of reinforcement learning. However, even in the cost-based formulation with its clear analogies to the scheduling problem, solely the cost of a state may have a clear interpretation. At the same time, it is difficult to formulate the expected cost-to-go for many useful objective functions in a way that an optimal problem substructure can be created and exploited. This may be done for particular
problem instances or for classes conforming to restrictive assumptions, but not, e.g.,
for the makespan in general. In addition, in problems with a continuous state space the
employment of numerical DP algorithms requires a discretization step, and therefore is
susceptible to exponential explosions. To remedy these issues, approximate dynamic
programming is postulated for certain optimal control and scheduling problems: see,
e.g., Lee and Lee [150]. Examples of application of a “pure” MDP formulation to
solving a scheduling problem (one with deadline and budget constraints) can be found
in, e.g., Yu, Buyya and Tham [272]. However, only the sequential branches of the
workflow are scheduled in this manner and the workflow must be partitioned for this
purpose; the part of the problem that is solved with the MDP-based approach (value
iteration, in this case) must have deterministic inputs and is solved statically.

2.12 Taxonomy in Scheduling for Parallel Processing

In the field of scheduling for parallel processing, given the “virtually infinite number of
problem types” in scheduling theory [145], Graham et al. [88] provide a classification,
later extended by Allahverdi, Gupta and Aldowaisan [9], based on a notation in the
form $\alpha|\beta|\gamma$. These literals encode, in order:

- $\alpha$: the machine environment (scheduling platform),
- $\beta$: the application model,
- $\gamma$: the objective function.

Allahverdi, Cheng and Kovalyov [10] later revised the taxonomy to include problems
with setup times and costs. Blażewicz, Lenstra and Kan [35], Brucker et al. [39]
and Herroelen, Demeulemeester and De Reyck [102] extended it to include Resource
Constrained Project Scheduling (RCPS) problems (see 2.2).

Each field contains literals for specific settings, e.g., $Q$ for a uniformly heterogeneous
machine environment or $prec$ for precedence constraints in the job model, and may
contain extensions of these. Tables 2.1-2.3 contain a non-exhaustive list of the literals
and extensions of relevance for this thesis and selected additional examples mentioned
in this chapter. For a complete list, see the works referenced above. No value for a
literal or extension indicates the default value: e.g., a missing \( prec \) reference indicates no precedence constraints.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Meaning and Extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>1</td>
<td>single machine</td>
</tr>
<tr>
<td></td>
<td>( P )</td>
<td>identical parallel machines</td>
</tr>
<tr>
<td></td>
<td>( Q )</td>
<td>uniform parallel machines</td>
</tr>
<tr>
<td></td>
<td>( R )</td>
<td>unrelated parallel machines</td>
</tr>
<tr>
<td></td>
<td>( * )</td>
<td>for any machine type ( * ), extensions are possible, e.g.: ( P2 ) for two identical machines, ( P\infty ) for an unbounded number of machines (if ( * = P ))</td>
</tr>
<tr>
<td>J</td>
<td>job shop</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>flow shop</td>
<td></td>
</tr>
<tr>
<td>PS</td>
<td>project scheduling</td>
<td></td>
</tr>
<tr>
<td>MPS</td>
<td>multi-mode project scheduling</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.1: Taxonomy in Scheduling for Parallel Processing: field \( \alpha \).

The standard notation is slightly expanded for the purposes of this thesis. For the sake of consistency, all such modifications are introduced in this section.

A value of \( * \) will be used in the following to indicate any value, e.g., \( *|prec|C_{\text{max}} \) is intended to refer to a makespan minimization problem, given a precedence-constrained job set to be scheduled onto any platform. More importantly, the introduction of problems involving queues requires minor additions to the range of \( \gamma \), i.e., an introduction of \( W \) and \( W_q \) as objective functions, with the meaning of total response time and in-queue waiting time, respectively. Additionally, the standard Kendall notation is used in Chapters 4 and beyond as a prefix to the (extended) Graham-Allahverdi code of the scheduling problem of interest, e.g., \( M/G/c : P| prec| E[C_{\text{max}}] \) for a queue of DAGs, arriving according to a Poisson process into the system, where the scheduling platform is composed of \( c \) identical, parallel processors.

In Chapter 5 Petri net and workflow net application models are introduced: extensions to the standard taxonomy referring to these models are proposed in Table 2.2. The Petri net model in general implies the presence of precedence constraints (\( prec \)), expressible with the simpler DAG model, but makes it also possible to express choice
and iteration, i.e., closed loops where activities may be repeated (see \[5.2\]). However, the communication links \((c_{ij})\) may be absent or present, hence they must be explicitly specified, if required. Optionally, closed loops may be also disallowed \((wn \backslash noloop)\).

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Meaning and Extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\beta)</td>
<td>-</td>
<td>loose job set (job shop or bag of tasks); no communication; no preemption</td>
</tr>
<tr>
<td></td>
<td>(prec)</td>
<td>precedence-constrained jobs (DAG)</td>
</tr>
<tr>
<td></td>
<td>(in-\backslash out-\backslash tree)</td>
<td>in-/out-tree-shaped DAG (implies (prec))</td>
</tr>
<tr>
<td></td>
<td>(wn\backslash wn \backslash noloop)</td>
<td>workflow net application model / workflow net with no loops (implies (prec))</td>
</tr>
<tr>
<td></td>
<td>(r_j)</td>
<td>problem with release dates</td>
</tr>
<tr>
<td></td>
<td>(c_{ij})</td>
<td>non-zero communication times (implies (prec) but requires it explicitly)</td>
</tr>
<tr>
<td></td>
<td>(v_i = k_i \div c_{ij} = k_c)</td>
<td>constant computation / communication times (e.g., unitary, as a special case)</td>
</tr>
<tr>
<td></td>
<td>(v_i, c_{ij} \sim stoch)</td>
<td>stochastic computation and communication times</td>
</tr>
<tr>
<td></td>
<td>(pmtn)</td>
<td>preemptive scheduling (at no cost); technically, not a feature of the application model but enforced as a member of (\beta) by the notation</td>
</tr>
</tbody>
</table>

Table 2.2: Taxonomy in Scheduling for Parallel Processing: field \(\beta\).

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Meaning and Extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>(\gamma)</td>
<td>(, \ldots)</td>
<td>objective(s) to be minimized: single item or list of comma-separated items (multi-objective)</td>
</tr>
<tr>
<td></td>
<td>(E[*])</td>
<td>expected value of *</td>
</tr>
<tr>
<td></td>
<td>(C_{\text{max}})</td>
<td>makespan (schedule length)</td>
</tr>
<tr>
<td></td>
<td>(\sum C_j)</td>
<td>sum of completion times (flowtime)</td>
</tr>
<tr>
<td></td>
<td>(\sum w_j C_j)</td>
<td>weighted sum of completion times</td>
</tr>
<tr>
<td></td>
<td>(W_q)</td>
<td>in-queue waiting time for queuing problems</td>
</tr>
<tr>
<td></td>
<td>(W)</td>
<td>total response time for queuing problems</td>
</tr>
</tbody>
</table>

Table 2.3: Taxonomy in Scheduling for Parallel Processing: field \(\gamma\).
Chapter 2. Background and Preliminaries

2.13 Offline and Online Scheduling Models

In Def. 2.18 (Sect. 2.4), an intuitive definition of online and offline policies was provided. This distinction naturally relates to the question when and how scheduling operations are performed: both hard constraints, such as feasibility, and soft ones, such as design choices, may influence the answer to that question. Van de Vonder, Demeulemeester and Herroelen [240] argued that the distinction between offline and online policies originates from the one between predictive and reactive scheduling paradigms. They also outlined an extended classification of such scheduling paradigms, which includes the special “mixed” case of proactive scheduling: in short, proactive scheduling has a baseline predictive behavior but switches to a reactive one when it is necessary or advantageous.

Traditionally, scheduling problems with deterministic inputs are mostly solved with the employment of offline (predictive) algorithms. Intuitively, if all the inputs are known a priori, there is no performance gain and thus no reason to “wait and see”, which would be a reactive approach. On the contrary, interleaving scheduling and execution may be a disadvantage, if, e.g., the scheduler is inefficient. On the other hand, for problems with stochastic inputs the use of online (reactive) policies is predominantly advocated: This is the case in stochastic scheduling (Pinedo [188] and of Möhring, Schulz and Uetz [173], Uetz [236] and Skutella and Uetz [221], see also Sect. 3.3). However, it is important to stress that neither of the input specifications strictly enforces any of the scheduling paradigms or models: for example, it may be a requirement that a problem with stochastic inputs be solved by an offline algorithm (see Chapter 3).

As pointed out by numerous authors, the existence of the reactive paradigm actually gives raise to not one, but a set of online scheduling models. Sgall’s thesis [216] proposed to distinguish “online” and “fully online” algorithms. Both differ, in Sgall’s classification, from offline algorithms because of the lack of complete knowledge on the inputs. In particular, for both online models, it is assumed that no knowledge on the jobs’ processing times is available until their completion. However, an “online” algorithm is provided with the dependency graph and the resource requirements in advance, whereas its “fully online” counterpart was denied this information as well. Pruhs, Sgall and Torng [194] extended the taxonomical notation of scheduling problems (see previous section) to include scheduling models identified in the previous work of Sgall.
2.13. Offline and Online Scheduling Models

[217], namely the “online-time”, “online-time-non-clairvoyant” and “online-list” models. In the first, the scheduler is unaware of the existence of a job until its release date, but at that moment is given foreknowledge of its (exact) processing time. In the second model, this knowledge is denied completely: not even probabilistic information is allowed. In the last model, each job’s processing time is known a priori and the next job in the list is revealed after each scheduling decision, i.e., without waiting for the next release date. In her thesis, Megow [169] identified further online scheduling models, later formalized and classified (for the weighted completion time problem) by Megow and Vredeveld [170]. In particular, a model was laid out, which had been previously implicit in the works of Feldmann et al. [77] and Azar and Epstein [20]. It is one where a scheduler learns about the very existence of a job when all its predecessors have completed their processing. At that moment, the job’s (expected) processing time, and all precedence relations to predecessors become known.

The referenced online models with different clairvoyance levels correspond to requirements posed by specific scenarios, as in [77] and [20] above. Sgall argued that the “non-clairvoyant” model represents very well the scheduling process in an operating system and the “list” model mirrors the workings of a load balancer. Both are essentially cases where job sets are generated at random, but, in some cases, jobs of particular types are well-known to the scheduler and, thus, have deterministic times.

In this thesis, workflow scheduling scenarios with more clairvoyance of the job set are investigated. Both when offline scheduling algorithms are required (Chapter 3) and when online policies are admitted (Chapters 4 and 5), it is assumed, and justified by the application scenarios presented, that the knowledge on the jobs dependencies (the task graph) is available at all times and is static. Unlike in Sgall’s work, there are no resource requirement considerations beyond those related to the allocation of tasks to processors (renewable resources) and time slots in these processors (non-renewable resources), i.e., to the model outlined in Sect. 2.3. The exact knowledge of future activity times, including job processing times is denied to all algorithms. In particular, in our formulation this concerns the offline schedulers working with stochastic inputs, which must provide a solution for any realization of the vectors of processing and communication times. On the other hand, the knowledge of the distribution of the activity times is available for all algorithms: this is what Megow and Vredeveld [170] termed a “stochastic online model”. Finally, the exact knowledge of the start and run times of completed and still running activities is assumed to be available in online policies.
Summarizing, the premise of this thesis is the clairvoyance of the activity set (tasks, inter-task communications) and its internal dependencies and of the stochastic variables describing the activity times. This assumption is central to the proposed solution approaches: the key point is to effectively exploit all available knowledge, including probabilistic information, on the inputs.

2.14 Optimization of Schedule Length

The adoption of heuristic approaches for most non-trivial parallel processing scheduling problems is motivated by a fundamental result due to the works of Ullman [238], Garey and Johnson [80] and Coffman, Jr., Garey and Johnson [54]. Namely, the deterministic makespan minimization problem $P|prec, c_{ij}|C_{max}$ is NP-hard for $|P| \geq 2$, $c_{ij} > 0$, $c_{ij} \neq 1$.

In other words, the makespan minimization problem is NP-hard for any non-trivial setting (arbitrary DAGs, non-zero communication edges, non-unitary task execution times) without preemption [62, 143], even with only two homogeneous processors. Therefore no polynomial-time algorithms for solving it exactly can exist, unless NP=P, see Karp [128] and Garey and Johnson [81]. This problem is obviously a special case of $\alpha|prec, c_{ij}|C_{max}$ with $\alpha \in \{Q, R\}$.

The above result summarizes how hard it is to solve a makespan optimization problem, even in a deterministic context with (multiple) homogeneous processors.

This has been the motivation for devising a great number of generic heuristic solutions, specialized and tailored heuristic algorithms and meta-heuristic approaches such as stochastic algorithms and linear (LP) or mixed non-linear programming (MNLP) approximations.

The class of deterministic makespan minimization problems does admit some easy problem subclasses, i.e., problems in $P$. Most of them require a homogeneous processing environment. Some easy problem classes are derived under either structural restrictions to the task graph (independent tasks, i.e., job shop, in the extreme case), or the value range or the relationship of the value ranges of the processing and communication times, or the number of processors, or a combination of these. For example, Jung, Kirousis and Spirakis [122] proposed an optimal algorithm for the makespan mini-
2.14. Optimization of Schedule Length

mization in a setting with an unbounded number of identical processors\(^3\) under the assumption of unitary computation times and a fixed computation-to-communication ratio, i.e., for the \(P\infty|\text{prec}, v_i = 1, c_{ij} = k|C_{\text{max}}\). They also argued that their algorithm provides a good measure of the complexity of an arbitrary DAG. Jakoby and Reischuk [119] proved the optimality of their algorithm for DAGs with \(k\)-ary trees and uniform communication delays and more complexity results for other tree structures and assumptions. Finally, some deterministic preemptive scheduling problems are easy, e.g., scheduling on two identical machines (see Muntz and Coffman, Jr. [176], Coffman, Jr. and Graham [55]). Gonzalez and Sahni [85] proved that the job shop deterministic problem in its preemptive version, i.e., \(Q|\text{pmtn}|C_{\text{max}}\) is in \(P\) even for any number of uniform machines, following McNaughton [168], who showed that the corresponding identical machine problem, i.e., \(P|\text{pmtn}|C_{\text{max}}\), can be solved by an algorithm of \(O(v)\) complexity.

The expected makespan minimization with stochastic inputs also admits easy problem classes, whereby the complexity is mostly unrelated to the deterministic “counterparts”. For example, Weiss and Pinedo [258] and Bruno, Downey and Frederickson [41] showed that a very simple policy, LEPT (Longest Expected Processing Times first), optimizes the expected makespan when scheduling sets of independent exponentially distributed jobs, i.e., solves to optimality the \(P|v_i \sim \text{Exp}(\lambda)|E[C_{\text{max}}]\) problem. As Weber [257] proved, the LEPT rule is optimal for jobs having a broader class of distributions: those with monotone failure rates. Interestingly, at the same time, it is known that the analogous problem with arbitrary deterministic times (\(P||C_{\text{max}}\)) is \(NP\)-hard for any \(|P| \geq 2\) [40].

Other examples of rules (rudimentary policies) which were derived, like LEPT, in the field of stochastic scheduling and do not have “deterministic” counterparts are briefly listed here. Pinedo and Weiss [190] demonstrated that the LVF rule (Largest Variance First, “with the same means” implied) is optimal for both the expected makespan and flowtime objectives. This is a very useful insight for any scheduling policy which must break ties between jobs that have the same, or approximately the same, priority ranking based on the expected durations. Chandy and Reynolds [46] showed that the HLF rule (Highest Levels First, where the level is simply the number of children) optimally

---

\(^3\)The setting with an unbounded number of identical processors (\(P\infty\)) is traditionally termed UNC (unlimited number of clusters), which is confusing and therefore avoided in this thesis.
solves $P|in\text{-}tree|E[C_{\text{max}}]$, i.e., the problem with in-tree precedence constraints. For the $Q||E[C_{\text{max}}]$ problem, i.e., the expected makespan minimization problem on a uniformly heterogeneous scheduling platform, Coffman et al. [56] proposed optimal rules for the cases $|Q| \in \{2, 3\}$, based on the concept of threshold rules introduced by Agrawala et al. in [2]. Again, no comparable results exist for the corresponding deterministic problem. Finally, Uetz [237] observed that, in a stochastic scheduling context in general, online list scheduling policies yield a better expected performances than any offline policy. This result extends a similar one by Kämpke [126], originally restricted to the case of exponential job distributions.

### 2.15 Evaluation of Schedule Length

A fundamental result is due to the works of Hagstrom [96, 97]: the problems of computing $P[C_{\text{max}} > d]$ and $E[C_{\text{max}}]$ are $\#P$-complete for $P_\infty|v_i \sim \text{stoch}, c_{ij} = 0, \text{prec}|$. A $\#P$-complete problem is one of counting the solutions of an NP-complete problem and, therefore, strictly harder than NP (Garey and Johnson [81]). Therefore, a scheduling policy that would rely on the exact determination of such a solution, would be computationally extremely expensive.

The result above describes the complexity of a problem originally formulated by Malcolm et al. [163] in the context of Program Evaluation Research Task (PERT: in the final report by Fazar [76], it was renamed into Program Evaluation and Review Technique, also conveniently abbreviated as PERT). In its basic form, it relates to the determination (exact or estimated) of the overall length of the execution of a project (task graph) encompassing precedence-constrained activities with probability distributions known a-priori.

Notably, the PERT problem is, strictly speaking, not a scheduling problem in the broad sense, as it does not entail an allocation problem. It is fairly obvious, however, that the corresponding scheduling problems are also $\#P$-complete for $|P| < \infty$ and for any of $Q \lor R|v_i \sim \text{stoch}, c_{ij} = 0 \lor c_{ij} \neq 0, \text{prec}|$. The rationale is that the complexity of the problem, according to the result reported above, is not related to the corresponding optimization problem, but to the evaluation of the solution.

The next subsection is dedicated to an in-depth discussion of this topic. Given the
complexity of the distributed schedule length computation problem, efficient estimation techniques are required, primarily not to evaluate the final solution, but to measure many partial solutions that may need to be evaluated during the scheduling process. In fact, any informed scheduling algorithm which must handle probabilistically distributed activity durations, needs to repeatedly evaluate the length of (partial) execution paths involving stochastic activities.

2.15.1 Related Work in Program Evaluation and Review Technique

Valdes [239] was the first to observe that if a PERT graph contains a subgraph where a parent and a child task have another common ancestor, their completion time distributions are correlated. Specifically, because of these particular precedence constraints, they become correlated at runtime, even if the random variables corresponding to the activities are assumed to be independent. He termed this set of precedence constraints an “interdictive graph”. The correlation is carried over to all child tasks of the interdictive graph. Naturally, this implies that exact makespan determination methods based on the assumption of independence cannot be applied to task graphs containing interdictive subgraphs. This includes convolution-based algorithms (see Sect. 2.9), which are regarded as acceptably “simple” in PERT problems. In other words, in cases with interdictive subgraphs, the PERT makespan computation problem requires direct integration of the full probability distributions.

For these reasons, various methods were proposed to eliminate interdictive subgraphs. For instance, in many approaches the common ancestor nodes were duplicated in order to create two pseudo-independent subgraphs (the term “node reduction” was sometimes used). Schmidt and Grossman [215], on the other hand, advocated the employment of Hsu’s PD-type functions. As they argued, these piecewise linear functions can approximate any activity duration distribution. At the same time, their application should result in an algorithm whose complexity is manageable. In fact, it was exponential only in the size of the largest irreducible subgraph with dependent paths. A more computationally efficient approach was later put forward by Ludwig, Mõh ring and Stork [160]. Li and Antonio [155] proposed a method to estimate the distribution of the makespan for a given schedule of a stochastic directed acyclic graph, based on the independence approximation criterion formulated by Kleinrock [137]. This method is
guaranteed to yield a rigorous overestimation of the actual makespan distribution and can yield the exact distribution in specific cases. However, it poses fairly restrictive requirements on the support limits of the probability distributions. It also remains too complex to be applied to evaluate a large number of stochastic DAG schedules.

2.16 List Scheduling

Among the most widely applicable approaches in classical scheduling, both for deterministic ([143]) and stochastic problems ([236]), irrespective of the objective function, is list scheduling. Its basic idea is that jobs be ordered into a priority list and, starting with the highest priority job, one by one assigned to the processor that allows for the earliest finish time of the job.

The minimization of the global makespan is an NP-complete problem even for deterministic DAGs (Sect. [2.14]): the global objective can only be optimized by exploring the whole state space. List scheduling is a generic heuristic approach that chooses to solve a set of related, tractable problems instead. First, a job priority list is determined, then, in a step-wise fashion, the job of highest priority which is still unscheduled is chosen and, for this job, the earliest finish time is minimized, across all possible job-to-processor allocations. In other words, the globally intractable makespan problem is transformed into a sequence of locally defined, polynomial problems of minimizing the earliest, or expected earliest finish time (EFT) of the job that has the highest priority at each step. The EFT is also only a heuristic in the generalized DAG scheduling problem and does not provide performance guarantees in general. The same is true for the many possible priority determination mechanisms. The priority list can be fixed before scheduling begins or recomputed at each scheduling step. As a final note, list scheduling has also been applied to problems other than makespan minimization, some of which may be also defined for job sets without precedence constraints, such as minimization of the weighted sum of completion times.
2.16. List Scheduling

2.16.1 Basic Techniques for Priority Determination

The most notable approaches to task prioritizing rely on so called levels, of which there are many kinds and variations. Levels are numerical attributes of task nodes in an annotated task graph, i.e., one where jobs and communication delays are assigned real-valued activity times. The levels may be computed based on a task graph-driven topological order or based on the relationship of the task to the critical path of the graph. The topological order may be top-down (t-level) or bottom-up (b-level). They may also be determined dynamically during a scheduling process. In this case, part of the information is based on the expected finish times of a subset of the tasks as determined by the scheduling decisions taken. The rest of the information is derived from the task graph.

In the following we discuss bottom (b-) and top (t-) levels (Gerasoulis and Yang [83]), dynamic levels (see Def. 2.51) and the dominant sequence attribute (see 2.50). The importance of these attributes stems from the fact that their variations are still in use in state-of-the-art algorithms.

Definition 2.46. The bottom level, or b-level, of a job (task node) \( j \) in a directed acyclic task graph with communication edges is the length of the longest path in the graph including \( j \) and any exit node and is given by:

\[
b(j) = w(j) + \max_{j_c \in J_{ch}(j)} \{b(j_c) + d(j,j_c)\}
\]

(2.43)

where \( w(j) \) is the intrinsic weight of \( j \), \( d(j,j_c) \) the intrinsic communication delay between the jobs \( j, j_c \) and \( J_{ch}(j) \) is the set of children of job \( j \) induced by the precedence constraints in the task graph.

The static b-level, or simply static level (SL), of a job \( j \) is its b-level computed under the assumption that the inter-task communication delays are set to zero:

\[
b(j) = w(j) + \max_{j_c \in J_{ch}(j)} \{b(j_c)\}
\]

(2.44)

Definition 2.47. The top level, or t-level, of a job (task node) \( j \) in a directed acyclic task graph with communication edges is the length of the longest path in the graph...
from $j$, excluding $j$ itself, to any entry node and is given by:

$$t(j) = \max_{j_p \in J_{\text{par}}(j)} \{ w(j_p) + t(j_p) + d(j, j_p) \} \tag{2.45}$$

where $w(j)$ is the intrinsic weight of $j$, $d(j, j_c)$ the intrinsic communication delay between the jobs $j$, $j_c$ and $J_{\text{par}}(j)$ is the set of parents of job $j$ induced by the precedence constraints in the task graph.

On heterogeneous scheduling platforms, the computation of bottom and top levels may additionally employ a normalization procedure, which takes into account the different speeds of the processors and communication channels in the platform. The classical example is the Heterogeneous Earliest Finish Time algorithm (HEFT), first proposed by Topcuoglu Hariri and Wu [233, 234], and many algorithms derived from, or inspired by it, such as SHEFT [227] (see also Sect. 3.3).

In its original version, HEFT is based on the average speed scaling strategy: an approach that has since been found sub-optimal ([274]). Other possible strategies include the maximum activity time on a platform, i.e., the slowest possible execution of a job or propagation of a job’s output, or the nominal, i.e., unnormalized activity speeds.

**Definition 2.48.** Let $(Q, Q \times Q)$ be a heterogeneous scheduling platform (see Sect. 2.2.2).

The downward rank, or speed-normalized b-level, of a job $j$ in a directed acyclic task graph with communication edges $(J, E)$ is the length of the longest path in the graph including $j$ and any exit node, adjusted by two speed factors $v_Q(j)$, $v_{Q \times Q}(j)$. It is given by:

$$b(j) = v_Q(j)w(j) + \max_{j_c \in J_{\text{ch}}(j)} \{ b(j_c) + v_{Q \times Q}(j)d(j, j_c) \} \tag{2.46}$$

The normalization strategy is termed average activity time, or average speed, or simply average, if the speed factor $v_Q(j)$ is the average speed of execution of any job $j$ on the platform and $v_{Q \times Q}(j)$ is the average speed of propagation of $j$’s outputs, i.e., if:

$$\forall j \in J \Rightarrow v_Q(j) = \avg_{p \in Q} \frac{w(j, p)}{w(j)} \tag{2.47}$$

where $w(j)$ is the intrinsic, or nominal, weight (execution time) of job $j$, $w(j, p)$ its
2.16. List Scheduling

Execution time when assigned to a processor $p$ and

$$\forall j \in J, j_c \in J_{\alpha}(j) \Rightarrow v_{Q \times Q}(j, j_c) = av_{Q_{\infty}Q} \frac{w(d(j, j_c), h)}{d(j, j_c)}$$  (2.48)

where $d(j, j_c)$ is the intrinsic, or nominal, duration of the communication delay between jobs $j, j_c$ and $w(d(j, j_c), h)$ the duration of this delay when the output of $j$ is transmitted along the channel $h$. In a uniform platform, $v_Q, v_{Q \times Q}$ defined as above are non-negative real-valued constants: in particular, they are independent of $j, j_c$.

The normalization strategy is termed maximum activity time (or minimum speed) if the speed factor $v_Q(j)$ is the minimum speed of execution of any job $j$ on the platform and $v_{Q \times Q}(j)$ is the minimum speed of propagation of $j$’s outputs. It is called unnormalized, or nominal, if the speed factors are assumed to be equal to one.

The upward rank, or speed-normalized t-level, of a job $j$ in a directed acyclic task graph with communication edges is the length of the longest path in the graph from $j$, excluding $j$ itself, to any entry node, adjusted by the speed factors $v_Q(j), v_{Q \times Q}(j)$, chosen according to any normalization strategy. It is given by:

$$t(j) = \max_{j_p \in J_{\text{par}(j)}} \{v_Q(j)p + t(j_p) + v_{Q \times Q}(j)d(j, j_p)\}$$  (2.49)

The idea of the critical path as a crucial concept in planning and scheduling has been recognized as early as 1959 by Kelley, Jr. and Walker [131, 130]. In the context of scheduling heuristics for parallel processing of directed acyclic task graphs, it has been applied starting with Kohler [138] and then Graham et al. [88] and Kasahara and Narita [129]. However, the most significant development of this method came with the dominant sequence, originated with the work of Erschler et al. [71] and successively applied as a priority list heuristic, first in a clustering algorithm by Kim and Park [136], then in list scheduling by Kwok and Ahmad [141] and Ahmad and Kwok [4].

**Definition 2.49.** The critical path (CP) in a directed acyclic task graph with communication edges is the set of directly connected activities that form the longest path between any entry task and any exit task in the graph. Given the set of exit nodes of the graph $J_{\text{exit}}$ (see Def. 2.14) and an initial value $J' = J_{\text{exit}}$, the set $J_{CP}$ of critical
path nodes is given by the following recursive equations:

\[
J_{CP}(J') = \{ j : \sup_{j \in J'}[\max_{j_p \in J_{par}(j)} t(j_p)] \}
\]  

(2.50)

\[
(J_{CP}, <) = (J_{CP} \cup J_{CP}(J'), <)
\]  

(2.51)

\[
J' = J_{par}(j)
\]  

(2.52)

Importantly, the set \( J_{CP} = (J_{CP}, <) \) is totally ordered.

Definition 2.50. Given a job set \( J \) and a critical path \( J_{CP} \) of a task graph as per Def. 2.49 the dominant sequence (DS) is a total order, over the entire job set \( J \), formed in the following way. The critical path nodes (CPNs) are processed in the order given by \( J_{CP} \). Each CPN that is either an entry node of the graph or has all of its parent nodes already in the DS, is immediately added to it. A parent node \( j_{IBN} \) of a CPN such that \( j_{IBN} \) does not belong to the CP, is termed an in-branch node (IBN). For any CPN, its IBNs that are not in the DS yet are recursively added to the DS, the IBNs with the highest b-levels first. In case of equal b-levels, ties are broken according to the lowest t-level. The CPN for which this set of operations has been performed, is added immediately after the required IBNs and the next CPN in \( J_{CP} \) is considered. Once all the CPNs and the associated IBNs are processed, the remaining task nodes, termed out-branch nodes (OBNs), are added to the DS, highest b-levels first.

Another set of experimentally successful scheduling policies is based on an attribute, the dynamic level (Sih and Lee [220]), which takes into consideration both the relationship of a job in a task graph to its critical path and the earliest start time of the job with respect to a processor pool. The latter is derived dynamically, at each scheduling step.

Definition 2.51. Given a directed acyclic task graph with communication edges and a scheduling process on a platform, the dynamic level (DL) of a job \( j \) on a processor \( p \) is the difference of the bottom level of \( j \) with respect to the task graph and \( S_{EST}(j, p) \), which is the earliest start time of job \( j \) on processor \( p \), obtained as per Eq. 2.7. Thus it is given by:

\[
dl(j, p) = b(j) - S_{EST}(j, p)\]  

(2.53)
2.16. List Scheduling

The earliest start and finish times of a job $j$ on a processor $p$ are a function of the decisions taken during the scheduling process, therefore, in general, they are different at each scheduling step. The information updated dynamically on the start and finish times of jobs may be used to update not just the DL, but also other task attributes. In particular, the actual finish times of the relevant activities (or their updated estimates) may be employed, if available, for the purpose of recomputing the t-level, as in the definition below.

**Definition 2.52.** The dynamic t-level of a job $j$ is its t-level computed by substituting the value of the finish time of any parent job $j_p$, if available, for the respective offline estimate. It is given by:

$$t(j) = \max_{j_p \in J_{par}(j)} \{C^*(j_p) + d(j, j_p)\}$$

(2.54)

where $C^*(j_p)$ takes the value of the finish time of the parent job $j_p$, if available, or of the offline estimate of the finish time of $j_p$, which is the sum of the top level and execution time of $j_p$ as in Eq. 2.45:

$$C^*(j_p) = \begin{cases} 
C(j_p) & \text{if } \exists C(j_p) \\
[w(j_p) + t(j_p)] & \text{otherwise}
\end{cases}$$

(2.55)

Based on the t-level, both the critical path and the dominant sequence can be re-evaluated (the updated finish times obviously have no relevance for bottom levels and static b-levels).

2.16.2 Notable Generic List Scheduling Algorithms

As early as 1966 Graham [86] published a generic list scheduling algorithm for the makespan problem, shown below. Notably, Graham’s algorithm, though originally targeted at an offline scheduling problem, can be applied in an online framework. Its performance depends on the priority list: different priority assignment mechanisms result in general in different performance, but there exists no analytical proof of the superiority of any of the priority mechanisms for any problem subclass or setting. In any case, in makespan minimization, the algorithm is $(2 - 1/|P|)$-competitive, where $|P|$ is the number of the identical processors available for scheduling. In other words,
Algorithm 1: A generic list scheduling algorithm with no deliberate idle times (greedy) due to Graham [86].

1. **input**: an instance of $P|r_j, prec|*; a job priority list $L$
2. **output**: a feasible schedule
3. $t \leftarrow 0 \triangleright time$
4. **while** $L$ has unscheduled jobs **do**
5.   **while** $L$ has unexamined jobs **do**
6.     $j \leftarrow$ first job in $L$
7.     **if** $j$ is available at $t$ and a processor is idle **then**
8.       schedule $j$ to any idle processor
9.       remove $j$ from the list $L$
10. $t \leftarrow t' > t'$ is the nearest moment in time when a processor is idle or a job is released, whichever occurs first

it has a guaranteed (worst-case) performance within a factor of $2 - 1/|P|$ of the optimal solution for both a $P|prec|C_{max}$ problem, as proved also by Graham [87], or $P|prec|E[C_{max}]$ (Chandy and Reynolds [46]). The performance guarantees given above are also known to be the best achievable for the cases $|P| \in \{2,3\}$, whereas for $|P| \geq 4$ no algorithm can be better than $(1 + 1/\sqrt{2})$-competitive (Faigle, Kern and Turán [75]). Interestingly, in all these cases the worst-case performance bounds are both valid and the best that can be guaranteed, irrespective of the priority list.

Variants of the algorithm with a priority list updated dynamically are naturally possible, see Graham [87] and Pinedo [189]: trivially, the priority list may be computed in the while loop, instead of being passed as an input to the above algorithm. Again, the performance guarantees referenced above are unaffected by the priority list employed and whether it is static or recomputed online. This argument, however, only holds for the worst-case performance bounds, not for the average or best-case performances, hence priority lists based on insights gained by the inspection of the task graph and, optionally, refined during the scheduling process are more successful than others. Another motivation for the development of such approaches is the fact that none of the above mentioned guarantees are applicable to problems with communication delays and non-identical processors.

An important shortcoming in Graham’s algorithm, which affects its average and achievable best performance is illustrated by the following reasoning. If the job with highest priority is not ready for execution, the next task in $L$ is scheduled. This policy does not insert idle times deliberately (it is also sometimes called non-idling in literature).
2.16. List Scheduling

Ultimately, the jobs are generally not scheduled in the order of L. The problem with this approach is that, if L is an optimal order for some objective function, the policy compromises this optimality in favor of short-term (local) idle time minimization.

To counter this effect, a similar policy which adheres strictly to the order imposed by the priority list was proposed by Munier, Queyranne and Schulz [175], shown in Alg. 2. It is referred to by some authors as job-based or deliberate idle time policy; in the following the term “lazy” will be used as in part of the subject literature, as opposed to “greedy”, used for Graham’s policy and policies derived from it. Some other authors, on the contrary, consider “greedy” those policies that strictly adhere to a fixed order and do not recompute priorities at runtime. This is a confusing naming convention that is not adopted here.

Algorithm 2: A generic list scheduling algorithm with deliberate idle times (lazy) due to Munier, Queyranne and Schulz [175].

```
1 input: an instance of P|rj,prec|\#; a job priority list L
2 output: a feasible schedule
3 t ← 0 ⊞ time
4 while L has unscheduled jobs do
5   while L has unexamined jobs do
6     j ← first job in L ⊞ only the highest priority job is considered
7     if j is available and a processor is idle then
8       schedule j to any idle processor
9       remove j from the list L
10     t ← t’ ⊞ t’ is the nearest moment in time when a processor is idle or a job is released,
11       whichever occurs first
```

In the work referenced above, Munier, Queyranne and Schulz also argued that a non-idling (greedy) policy is appropriate for optimizing objectives related to machine utilization, including the makespan, whereas a job-based (lazy) one is better suited to optimize job-centric objectives like the sum of weighted completion times. This argument is valid in the sense that a greedy policy tends to maximize resource utilization (keep processors busy), while Graham’s result [87] guarantees that any greedy policy has a worst-case performance exactly equal to a lazy one (or any other list scheduler). However, as discussed above, this reasoning applies only to the worst-case performance and to problems with no optional communication delays. If such delays are permitted, as in all the problems examined in this thesis, a greedy algorithm is not a good candidate, no matter whether the problem has deterministic or stochastic activity.
times (and a makespan or expected makespan objective). Summarizing, the argument formulated by Munier, Queyranne and Schulz on the near-optimality of greedy algorithms for makespan problems is actually contradicted by the experimental success of many list scheduling heuristics, based on different priority ordering mechanisms. This success, clearly, reflects average performances. Most of these heuristics in fact do follow their priority list strictly in the successive scheduling steps. A few jobs eventually appear to be effectively scheduled “out of order”, but only under very specific circumstances. In particular, this may happen due to an insertion into an idle time slot of a task of lower priority, after the algorithm ensures that a job of higher priority, with respect to the prescribed order, will not be delayed. In other words, the insertion is performed only if and only as long as the idle time slot is not used up, which in the following will be termed “safe insertion”. Conversely, an algorithm that allows for greedy scheduling of low priority jobs with the safe insertion approach only will be called “safely greedy”.

There exist several implementations of this principle. The Modified Critical Path (MCP) by Wu and Gajski [267], on offline algorithm for the $P|\text{prec},c_{ij}|C_{\text{max}}$ problem, simply finds the idle time slot that minimizes the finish time of a job, if an insertion is feasible and advantageous. Kruatrachue’s Insertion Scheduling Heuristic (ISH, [139]) first schedules the job of highest priority without attempting an insertion, then tries to insert as many of the remaining jobs into the idle time slot thus generated, as long as this minimizes their finish times. Another notable algorithm, the Heterogeneous Earliest Finish Time (HEFT) by Topcuoglu, Hariri and Wu [233] follows the more straightforward insert-first approach of MCP to solve the corresponding scheduling problem on a uniformly heterogeneous platform ($Q|\text{prec},c_{ij}|C_{\text{max}}$). For a generic
illustration of this method, see Algorithm 3.

**Algorithm 3:** A generic offline safely greedy list scheduling algorithm for the makespan objective ($\gamma \in \{C_{max}, E[C_{max}]\}$).

```plaintext
1 input: an instance of $*|r_j, prec|\gamma$; a job priority list $L$
2 output: a feasible schedule
3 while $L$ has unscheduled jobs do
4     $j \leftarrow$ first job in $L$ \> the highest priority job in the list
5     $p \leftarrow$ the processor that minimizes the earliest finish time of $j$ with insertion into idle times
       if constraints are respected and other tasks need not be shifted
6     schedule $j$ to $p$
7     remove $j$ from the list $L$
```

This principle can also be applied in an online scheduler. The idea of a safe insertion remains the same: at a given time, a job of a lower priority may be scheduled (“inserted”), if it will not interfere with the earliest finish-time mapping of a job of higher priority. This reformulation, shown in Alg. 4, is used whenever the safe insertion approach is applied in an online policy, e.g., when a version of HEFT is ported to an
online problem setting.

**Algorithm 4:** A generic online safely greedy list scheduling algorithm for the makespan objective ($\gamma \in \{C_{\text{max}}, E[C_{\text{max}}]\}$).

```plaintext
1 input: an instance of $\star |r_j,\text{prec} |\gamma$; a job priority list $L$
2 output: a feasible schedule
3 $t \leftarrow 0 \triangleright$ time
4 $M \leftarrow \emptyset$ \triangleright set of higher priority partial schedules
5 while $L$ has unscheduled jobs do
6     while $L$ has unexamined jobs do
7         $j \leftarrow$ first job in $L$ \triangleright the highest priority job in the list
8         $M(j) \leftarrow \{\}$ \triangleright set of all partial schedules for $j$
9         foreach processor $p$ do
10             $M(j) \leftarrow M(j) \cup \{(j, p, ST, FT)\}$ \triangleright cache the earliest finish-time mapping of $j$ on $p$ if it does not overlap with any member of $M$
11             $m = (j, p, ST, FT) \leftarrow$ the mapping in $M(j)$ for which the earliest finish time of $j$ is minimal
12         if $j$ is available and $a_p$ is idle then
13             schedule $j$ to $p$
14             remove $j$ from the list $L$ and the associated element in $M$, if any
15         else
16             cache $m$ in $M$
17     $t \leftarrow t'$ \triangleright $t'$ is the nearest moment in time when a processor is idle or a job is released, whichever occurs first
```

As discussed, offline greedy algorithms can deliver suboptimal solutions. This is also true in many online stochastic scheduling problems. While investigating such problems with a weighted completion time objective, Uetz [237] observed that greediness fails irrespective of the priority list and proved this fact for other objective functions that are regular and additive (see 2.6), such as the mentioned sum of weighted completion times. However, there is a middle ground between greedy and lazy schedulers. Chekuri et al. [48] presented an algorithm for offline (or online-time clairvoyant, see 2.13) scheduling of jobs to minimize the total weighted completion time, shown in Alg. [5]. It is particularly interesting because it effectively employs a parameter-driven mechanism which balances the idle time accumulated and the amount of jobs scheduled “out-of-order”.

Notably, this algorithm is also applicable for stochastic scheduling problems with an
optimization objective defined as the expectation of a function of stochastic variables (e.g., expected makespan): in that case, expected values of the accumulated idle times are used (Line 15).

Algorithm 5: The Accumulated Idle Time Charging algorithm (ITC, Chekuri et al. [48]).

1. \textbf{input:} an instance of $P|r_j, prec|*$; a job priority list $L$; a threshold parameter $\beta$
2. \textbf{output:} a feasible schedule
3. $t \leftarrow 0$ \triangleright time
4. $L_\beta \leftarrow \{\} \triangleright a list of accumulated idle times per job
5. \textbf{while} $L$ has unscheduled jobs \textbf{do}
6. \hspace{1em} \textbf{while} $L$ has unexamined jobs \textbf{do}
7. \hspace{2em} $j \leftarrow$ first job in $L$
8. \hspace{2em} \textbf{if} $j$ is available at $t$ and a processor is idle \textbf{then}
9. \hspace{3em} \textbf{if} $j$ is the highest priority unscheduled job \textbf{then}
10. \hspace{4em} schedule $j$ to any idle processor \triangleright the highest priority job is scheduled
11. \hspace{4em} unconditionally
12. \hspace{4em} remove $j$ from the list $L$
13. \hspace{2em} \textbf{else}
14. \hspace{3em} \textbf{if} there is no value for $j$ in $L_\beta$ \textbf{then}
15. \hspace{4em} add $t$ as starting moment of accumulated idle time for $j$
16. \hspace{3em} \textbf{else}
17. \hspace{4em} \textbf{if} the (expected) value for $j$ in $L_\beta$ exceeds $\beta$ \textbf{then}
18. \hspace{5em} schedule $j$ to an idle processor and remove it from $L$ \triangleright otherwise, job $j$ will keep accumulating idle time until it exceeds the threshold $\beta$
19. \hspace{2em} $t \leftarrow t'$ \triangleright $t'$ is the nearest time when a processor is idle or a job is released, whichever occurs first

Another classical algorithm of generic applicability is Sih and Lee’s dynamic level scheduling (DLS [220]), an offline makespan-minimizing heuristic, based on the dynamic level attribute, discussed in [2.53] originally targeted at uniformly heterogeneous platforms and since then repeatedly modified and applied, e.g., in cloud environments ([256]) or, under the assumption of normal distributions [153], to a restricted class of stochastic workflows. In addition employing a different priority attribute, DLS evaluates all task-processor pairs instead of candidate processors for each task. As such, it is structured differently to the list schedulers reported above, hence an outline is provided in Alg. [6] It is worth noting that, like the plain greedy algorithm, DLS does not admit
deliberate idle times. A version of DLS for an online scheduling scenario can be easily derived in analogy to what has been illustrated above (Alg. 3-4).

**Algorithm 6:** The dynamic level scheduling algorithm (DLS, Sih and Lee [220]).

```plaintext
1 input: an instance of $Q|c_{ij},\text{prec}|C_{\text{max}}$; a job priority list $L$ static bottom level-based by default
2 output: a feasible schedule
3 while $L$ has unscheduled jobs do
4   foreach processor $p \in Q$ do
5     foreach job $j$ available on $p$ at the current step do
6         determine the dynamic level of $j$ on $p$ \(\text{Eq. 2.53}\)
7     find the job-processor pair $(j, p)$ with the maximum dynamic level
8     schedule $j$ on $p$ at the earliest start time
9     remove $j$ from the list $L$
```

2.17 Uncertainty and Robustness in Scheduling

Ierapetritou and Li [113] provided an extensive survey from a recent perspective and a historical outlook on the general problem of modeling and managing uncertainty in various domains.

Originally, the effects of uncertainty, understood as unpredicted disruption or perturbation, on previously fixed production plans and schedules drew the attention of the researchers to this problem. Graves [89] pointed out that schedules obtained from practical experimentation often differ substantially from theoretically planned ones, even in the presence of only slight disturbances in the processing times of the jobs. He also suggested to include uncertainty explicitly in the scheduling process. Subsequent investigation included the estimation of perturbation effects, as in He, Smith and Dudek [101], Lawrence and Sewell [146] or Lejmi and Sabuncuoglu [151]. Countermeasures were also proposed, such as re-scheduling, as in, e.g., Abumaziar and Svestka [1] and again Lawrence and Sewell [146], but also employment of reactive scheduling, as proposed by Sabuncuoglu and Kizilisik [207] for flexible manufacturing systems.

Successively, the research on a systematic estimation of the effects of the perturbation resulted in the elaboration of a series of metrics that quantify the robustness of a schedule. Informally, the robustness of a schedule indicates its resistance to a potentially
destructive disturbance, or disruption, such as a machine failure, or to a perturbation in the processing times of a set of jobs. It may also be generalized as the degree to which the disruption or perturbation in the inputs (job and communication times) is carried over to the output (a precomputed schedule). An early proposal of a robustness metric is due to Leon, Wu and Storer [152], who conceived of the post-disturbance makespan and post-disturbance makespan variability as of two components of such a metric. A precise definition of schedule robustness is clearly correlated with the metric of choice. Moreover, a schedule robustness metric does not have to be makespan-based, although such metrics are predominant. Leon, Wu and Storer, besides robustness metrics, also proposed countermeasures to handle disturbances, such as the right-shift control policy, which delays the incomplete jobs to preserve the schedule.

Bölöni and Marinescu [38] introduced two new robustness measures: the number of critical components within a schedule and the schedule entropy. For both of these measures higher values indicate potentially less robust schedules: Specifically, a higher entropy entails a higher probability that more than one of a set of paths within the schedule may become its critical path.

In an extensive survey, Canon and Jeannot in [44] examined several metrics directly related to the schedule length: makespan standard deviation, differential entropy, average slack, slack standard deviation, average lateness and a bound-based probabilistic metric. They concluded that even the most accurate and computationally most expensive candidate metric examined, i.e., the bound-based probabilistic metric, does not seem to provide more useful information than the simplest one, which is the standard deviation of the makespan.

The makespan standard deviation is the robustness metric used in the experimental analysis of 20 static DAG scheduling heuristics for the $Q|\text{pre}c, c_{ij}|C_{\text{max}}$ problem, conducted by Canon et al. [45]. Notably, the authors did not attempt to solve the corresponding expected makespan or makespan standard deviation minimization problem directly. The aim was to measure the robustness of the available solutions, computed by offline heuristics for the deterministic problem, when the actual execution time of the tasks differ, due to perturbation, from the values the original algorithms were given.

The most notable conclusion of the study is that there exists a correlation between the value of the makespan and its robustness. From the specific point of view of the minimization of the makespan standard deviation the following findings are the most impor-
Chapter 2. Background and Preliminaries

tant. Firstly, Canon et al. were able to prove that “simple rules are quite robust under both stochastic disturbances and variations in processing times and due dates”, in particular, longest expected processing time first (LEPT) and largest variance first (LVF). Secondly, as already observed by Sakellariou and Zhao [209], they argued that scheduling heuristics that perform well with deterministic inputs are the most robust ones. In particular, the following heuristics were found to be the best for both makespan and robustness: HEFT [233], HBMCT [208], DLS [220] and PCT [165]. Canon et al. in [45] also noted that, if at runtime a set of inputs is given which are realizations (samples) of stochastic vectors, it is not possible to keep the precomputed schedules based on the static estimates of these stochastic vectors. Trivially, the actual start and finish times in the sample-adjusted schedules will differ from the tentative ones. In other words, an offline solution based on the estimates is not a solution to the scheduling problem in the strict sense. The authors additionally considered a scenario termed “assignment strategy” in which, while the job-to-processor assignment resulting from the precomputed schedule is fixed, the order of the execution of the tasks on any processor may be changed, if feasible for a particular realization. A pure assignment problem does not imply strict sequencing, as it does not imply strict timing. In this respect, the final and surprising finding was that allowing for the runtime task order adjustment, i.e., the “assignment strategy” results in a performance inferior to the one of the “sequence strategy”. The latter, conversely, retains the order implied by the precomputed schedule across all realizations of the stochastic DAG.

The works referenced above contain relevant research in robustness metrics and their application to classical scheduling algorithms working with deterministic values. However, scheduling algorithms can obviously be designed with robustness requirements (or constraints) in mind and the design process may directly incorporate these requirements.

Bölöni and Marinescu [38] analyzed the schedule entropy and the impact of the tasks outside of the most likely critical path. They proposed heuristics that are able to determine more robust solutions among the set of schedules having the same length. Gupta, Parmentier and Trystram [94] proposed the addition of artificially inserted edges (“pseudo-edges”) to preserve the original schedule and, specifically, to prevent the disruption of the originally planned execution of jobs of high priority by random processing time changes. Effectively, theirs was a partially online scheduling policy
that enforces a strict order in accordance with a precomputed priority list, despite the presence of runtime disturbances. It did not fully take advantage of the information becoming available at runtime, which would require rebuilding the schedule. A similar approach was developed by Guinand Moukrim and Sanlaville [93]. While including supplementary constraints to guarantee robustness, it also advocated the sensitivity analysis method as a means of incorporating uncertainty. It took advantage of deterministic optimal algorithms known for problem settings with tree-like task graphs with unitary communication times and thus was applicable only to such settings. Shi, Jeannot and Dongarra [219] used two different definitions of robustness based on tardiness and miss rate. The latter is the ratio of schedule realizations that have a larger makespan than the offline schedule, computed based on expected activity durations, to the total number of generated schedule realizations. They formulated the robustness problem as a bi-criteria optimization problem: to minimize the makespan while maximizing the total schedule slack, which is the sum of the slacks of all jobs. To solve the problem, a genetic algorithm was proposed, which in turn employed the $\epsilon$-constraint method. Shestak et al. [218] outlined a mathematical model for schedule robustness based on confidence intervals. Although their model was potentially applicable to a generalized QoS robustness, in practice, the solutions put forward focus on the makespan. Specifically, the robustness problem was formulated according to the confidence interval model, as the single-objective problem of finding the schedule that maximizes the probability that the makespan will be confined in a certain interval. To solve it, four list scheduling heuristics were proposed, using each a different amount of information derived from the probability distributions of the stochastic variables of interest. In addition, a range of stochastic algorithms was also examined, including a genetic algorithm and variants of the ant colony optimization and simulated annealing meta-heuristics.

Ali et al. [8] investigated the problem of scheduling a continuous inflow of heterogeneous applications onto a heterogeneous platform. The primary target of their system was to provide guarantees that specific QoS constraints, a minimum throughput level and a maximum end-to-end system latency, are satisfied. Given these assumptions, the problem was defined as designing a set of offline resource allocation heuristics that are intrinsically robust, while allowing for runtime reallocations, if needed. An offline allocation was defined as robust if it maximized the allowable workload increase before a reallocation was triggered to avoid a QoS constraint violation. Another goal was to
minimize the failure rate of the employed heuristic, i.e., the number of instances for which the heuristic could not find a solution satisfying the original QoS constraints. A genetic algorithm and a variant of simulated annealing were proposed as a solution to these problems.

Sabuncuoglu and Goren [206] provided a taxonomical view on proactive, reactive and other approaches to handle uncertainty and guarantee robustness of schedules, abstracting from specific scheduling scenarios.

Mahjoub, Sánchez and Trystram [162] proposed another robust algorithm designed to alleviate the effects of disturbances in the inputs on the final, effective schedule length. The performance metric employed was the “stability ratio”, defined as the quotient of the effective makespan obtained under disturbances and the original makespan, computed initially in an offline fashion under the assumption of deterministic inputs. The proposed RCCA algorithm was based on the idea of clustering.

Leon, Wu and Storer [152], made the following important observation about the stochastic scheduling approach. Considering uncertainty explicitly in stochastic scheduling as suggested by Graves [89] actually avoids most of the ramifications of uncertainty by employing an online reactive approach. In stochastic scheduling there is no a priori commitment to an entire schedule whose robustness could be measured. An online policy is triggered repeatedly throughout the execution process of the set of jobs to be scheduled. When the policy is triggered at a time $t$, it is assumed to know which jobs have finished up to $t$ and when and on which processor. Thus the uncertainty, from the perspective of an online policy, is limited to the set of the jobs that are still being executed or have not been scheduled yet. More importantly, at no time $t$ is an online policy required to commit to any future partial schedule, i.e., that a job $j$ is bound to be started at some definite time $t' > t$. Leon, Wu and Storer [152] argued that all of these factors make the robustness problem easier in the stochastic scheduling approach, understood as online scheduling with built-in uncertainty.

### 2.18 Queuing Theory

Another field of research related to scheduling and relevant for the scope of this thesis investigates queues, originated with the work of Erlang in applying probabilistic mod-
els to servicing telephone conversations \cite{70}. Asmussen and Boxma \cite{17} provide an account of the historical development of queuing theory (QT). Gross et al. \cite{90} give an updated, systematic view on the field. The importance of QT may be summarized as follows. For a class of models that represent any kind of service facility with multiple processing entities and a customer arrival process, queuing theory investigates macroscopic properties such as capacity or stability, while abstracting away most details concerning individual system states. From the model’s perspective, customers requiring service do not differ from jobs requiring processing and are not distinguishable individuals (although they may belong to different classes, see below).

**Definition 2.53.** A queue is a stochastic process whose state space is the set of natural numbers $\mathbb{N}$. The state (value) assumed by the process represents the number of jobs (or, in the classical formulation, customers) in the system. The jobs (customers) are indistinguishable and unrelated atomic entities, unless otherwise specified. A queue is associated with a number $c \in \mathbb{N}_+ \cup \{\infty\}$ of processors, or servers. These are assumed to be identical and serve from the front of the queue, unless otherwise specified. At any time, if there are more jobs in the system than processors, the jobs not (yet) assigned to a server queue in a buffer, whose size is given by $k \in \mathbb{N}_+ \cup \{\infty\}$.

A queue is associated with additional, discrete-time stochastic processes: an arrival process of jobs into the queue, a service process and a departure process of jobs that have undergone processing.

The processes in the above definition may have a arbitrary distributions. It is usually assumed, however, that the arrival and service time distributions belong to a known distribution class and that the average arrival rate $\lambda$ and average service rate $\mu$ are known.

The standard notation for taxonomical classification of such settings is due to Kendall \cite{133} and takes the form: $A/B/C$, where $A$ represents a code for the arrival process, $B$ the service time distribution and $C$ the number of servers, respectively. This notation has later been extended by Lee \cite{147} to include $K/D$ (system capacity and queue discipline) and by Taha \cite{226}, to specify the size of calling population ($N$). Table 2.4 shows parts of the notation of relevance for this thesis.

Clearly, the total time spent by a single job in the queue, from arrival to departure, is the sum of its in-queue waiting time, which may be zero, and its service time. In
Table 2.4: Kendall’s notation with Lee’s and Taha’s extensions: selected values.

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>A</strong></td>
<td></td>
<td>Arrival time distribution:</td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>Poisson / memoryless</td>
</tr>
<tr>
<td></td>
<td>G/GI</td>
<td>general distribution / general distribution with independent arrivals</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>degenerate distribution: deterministic / fixed arrival times</td>
</tr>
<tr>
<td><strong>B</strong></td>
<td></td>
<td>Service time distribution:</td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>Poisson</td>
</tr>
<tr>
<td></td>
<td>G</td>
<td>general distribution</td>
</tr>
<tr>
<td></td>
<td>D</td>
<td>degenerate distribution: deterministic / fixed service time</td>
</tr>
<tr>
<td><strong>C</strong></td>
<td>1</td>
<td>Number of servers:</td>
</tr>
<tr>
<td></td>
<td>c/k</td>
<td>single-server</td>
</tr>
<tr>
<td></td>
<td></td>
<td>multiple (identical) servers</td>
</tr>
<tr>
<td><strong>K</strong></td>
<td></td>
<td>System capacity (infinite if omitted)</td>
</tr>
<tr>
<td><strong>D</strong></td>
<td>FIFO/FCFS</td>
<td>Queuing discipline:</td>
</tr>
<tr>
<td></td>
<td>SIRO</td>
<td>first-in, first-out (first-come, first-served)</td>
</tr>
<tr>
<td></td>
<td>PS</td>
<td>service in random order</td>
</tr>
<tr>
<td></td>
<td></td>
<td>processor sharing</td>
</tr>
<tr>
<td><strong>N</strong></td>
<td></td>
<td>Calling population / maximum number of customers or jobs (infinite if omitted)</td>
</tr>
</tbody>
</table>

general, the rate and distribution of the departure process, which describes this total time for the entire calling population, depends non trivially on the arrival and service time distribution, as well as on the queuing discipline.

**Definition 2.54.** The expected response time of the system is the mean time spent in the queue by a job and it is given by:

\[
E[W] = E[W_q] + E[S]
\]  

(2.56)

where \( E[W_q] \) is the mean waiting time and \( E[S] = \frac{1}{\mu} \) is the mean service time.

The utilization, or load of the system is given by:

\[
\rho = \frac{\lambda}{\mu}
\]  

(2.57)

The definitions above assume \( \mu \) as the effective service rate of the queue, taking into account all servers, in contrast to some authors who prefer the notation \( c\mu \), to stress
that there are \( c \) identical servers, each with a rate of \( \mu \).

Two results are of primary importance for this thesis, as they are applicable to all queues, irrespective of the associated arrival and service time distributions.

Kendall [132] showed that in any queue with an infinite calling population the number of jobs waiting for service, also termed size of the backlog, grows to infinity unless the utilization is strictly smaller than one. As a consequence, the waiting time and thus the expected response time grow to infinity. Kendall’s result is also known as stability condition: A queue is stable if and only if the expected response time is bounded. An exception is the trivial case of a \( D/D/c \) queue, in which a utilization not greater than one is a sufficient stability condition.

Little’s law, conjectured by Cobham [53] and proved by Little [158], states that in a long-running queue the expected number of jobs \( E[N] \) in the system is the product of the average arrival rate \( \lambda \) and the expected response time \( E[W] \) of the system.

### 2.19 Experimental Evaluation

The experimental performance evaluation methodology applied in this thesis is best outlined in the relation, but also in contrast to, the competitive analysis (CA) method. This method was first proposed by Sleator and Tarjan [222] for the problem of list update and paging rules, with the actual term introduced in Karlin et al. [127] for the related caching problem. The definitions were refined several times, e.g., by Fiat et al. [78]. An explicit outline of the standard procedure to evaluate the performance of online algorithms by means of CA was given by Albers [6]. It employs the concept of approximation, defined below.

**Definition 2.55.** An algorithm \( \pi \) is an \( \alpha \)-approximation algorithm of an algorithm \( \pi^{\ast} \) if, given a constant \( \alpha \in \mathbb{R} \), for any input \( \kappa \), the following holds:

\[
\pi(\kappa) \leq \alpha \pi^{\ast}(\kappa)
\]  

(2.58)

If a problem \( P \) admits an approximation algorithm, it is called approximable.

Approximation is related to the bound of the performance, as noted by Błażewicz et al. [34]. Strictly speaking, the approximation factor with respect to a reference
(usually, the optimal offline or online algorithm) gives the guarantee on lower bound of performance of the examined algorithm. However, the tested algorithm can still perform better with some (or even infinitely many) problem instances. Błażewicz et al. ([34], [33]) also defined the closely related concept of approximation algorithm as a heuristic algorithm with an analytically evaluated accuracy.

Albers [6] linked the two concepts discussed in this section by stating that the closer the (guaranteed or experimentally derived) approximation of the optimal solution is, the more competitive the tested algorithm is. In the methodology (proposed for online algorithms), however, Albers assumed that the reference must be an optimal offline algorithm, which is given the whole set of inputs in advance. In an analogous manner, for the purpose of evaluating stochastic scheduling policies, Skutella and Uetz [221] assumed that the online optimization algorithm $\pi$ whose performance is measured must be non-anticipatory (i.e., must not have any knowledge of future events). They proposed to measure its competitive ratio as a factor of the performance of the optimal fully informed algorithm $\pi_{\text{oracle}}$.

In the experiments conducted throughout this thesis, whether the performance of offline or online algorithms is evaluated, it is not a substantial issue to provide the reference algorithm with complete knowledge of the actual realizations of all the inputs, i.e., activity times and arrival times of workflow instances. For a finite number of instances, the information needed by $\pi_{\text{oracle}}$ is readily available. The actual issue is that $\pi_{\text{oracle}}$ would be required to solve a large number of NP-hard problems.

As this is impractical, throughout the experimental evaluation sections in this thesis, a method is followed, that is a generalization of the procedure originally proposed by Ahmad and Kwok [3] and further expanded by Kwok and Ahmad [142].

These authors employed an approximate measure of performance termed normalized schedule length (NSL), which is the quotient of the makespan obtained using any of the tested algorithms and of the length of the critical path of the considered task graph. The NSL may be seen as computationally simpler equivalent of the competitive analysis in the context of the $P|\text{prec, } c_{ij}|C_{\text{max}}$ (i.e., the deterministic makespan) problem.

In an analogous manner, in the experimental evaluation carried out throughout this thesis, [5,6], for each policy $\pi$ and objective $\gamma$, a performance measure is computed according to the following procedure. For each workflow instance, the assignment-
independent critical path is computed, which may be different due to the different realizations of the activities. Similarly, for each instance, the value of the objective $\gamma$ is determined: in this thesis, this is the makespan or the response time. The quotient of these two values is the normalized value of $\gamma$ (e.g., the normalized total response time $E[W_n]$) for the given instance. Successively, for all the $N$ (e.g., 100) instances in the experiment, the average value among these quotients is derived, see equation below. It is employed as the performance measure for the scheduling policy of interest.

$$E[\gamma_n](\pi) = \frac{1}{N} \sum_{i=0}^{N} \gamma_n(i, \pi) = \frac{1}{N} \sum_{i=0}^{N} \frac{\gamma(i, \pi)}{\sum_{j \in J_{CP}(i)} r(i, w(j))}$$ (2.59)

This is the expected normalized value of $\gamma$: Strictly speaking, it is not the true expected value, but an approximation, whose accuracy depends on the number $N$ of samples. In other words, in practice it is also true that $E_n[\gamma](\pi) = E_n[\gamma](\pi)(N)$. However, in all practical simulations a large number $N$ of samples are generated. Thus, it may be assumed that the sample-based metric converges to the true mean.

Apart from the average of the normalized value of $\gamma$ (which approximates the expected value in the outlined formulation), it is naturally possible and straightforward to compute and visualize statistical dispersion measures, such as the descriptive five-number summaries (see Hoaglin, Mosteller and Tukey [104]), also shown in the figures throughout the evaluation sections in this thesis. In addition, an equation analogous to the above Eq. (2.59) is applied for robustness measures, relative to the standard deviation of the values of the objective function $\gamma$ (see Sect. 2.17).

### 2.20 Summary

A scheduling problem in the strict sense concerns the timing of arrival and departure of units of work requiring processing or service. In a broader sense, scheduling also entails allocation, which is the assignment of resources to a number of entities which require them for an action, and sequencing, which pertains to the order of the processing. Further, the problem of scheduling jobs arriving at random times into any processing system also implies managing waiting lines. This thesis focusses on a number of problem classes that combine allocation, scheduling, sequencing and, with the
Chapter 2. Background and Preliminaries

Exception of Chapter 3 waiting line problems. The underlying scheduling platform is assumed to consist of a set of processors, available in parallel in a system targeted at allocating and processing a set of applications, and of a set of physical communication channels connecting these processors. The basic application model, accordingly, is assumed to be formed by a precedence-constrained set of atomic jobs, or tasks. In this model, any precedence constraint between two jobs may be mapped to an optional communication delay: If the jobs are scheduled to different processors, this is the time needed to transfer working data between these jobs, using the communication channels which connect the corresponding processors. We refer to the jobs and communication delays using the umbrella term “activities”. In all problems investigated in this thesis, we assume that the intrinsic execution times of all activities, i.e., their runtimes at the nominal speed, are described by arbitrary probability distributions, which are known. When the activities are executed on the underlying platform, their effective runtime is a function of the speed of the platform’s resources (processors and communication channels) the activities are assigned to: These speeds are known deterministic values.

Given these premises, in general, we want to find an algorithmic method, called a scheduling policy, which solves the problem of determining a schedule: an allocation of all the jobs in the application to a set of time intervals defined on the processors available in the scheduling platform. Naturally, the focus is on feasible schedules, which respect both the application-level precedence constraints and the additional constraints introduced by inter-task communication delays. Any solution (schedule) can be evaluated with respect to a set of objective functions, which usually measure the consumption of non-renewable resources, such as elapsed time, or utilization of renewable resources (processors): The objectives are termed global if they are defined for the entire solution. Conversely, any policy can be evaluated with respect to a given set of objective functions and a set of inputs, such as activity runtimes or available processing resources: For a set of inputs, we obtain the corresponding performance space of the policy. Given a performance space, the expected performance of the policy is computable for any objective. Additionally, a policy for which this expectation is optimal always exists for any problem in which the probability distribution of the inputs is finite discrete or has a Lebesgue density. With a set of global objective functions, scheduling can be formulated in general as an optimization problem: in particular, as a sequential decision process, where at each step a decision is made with respect to a set of locally defined objectives, related to the global objectives, but not coinciding
with them. For example, it can be regarded as a Markov decision process in which, at any step, the expected cost over a given number of future steps (horizon) is minimized. This formulation gives valuable insights to understand the complexity of the schedule optimization problem, especially for particular objectives related to execution time, and is at the root of dynamic programming and stochastic scheduling approaches.

This thesis investigates a range of problems in which the objective functions describe the total time spent by the application in the system, or total response time. The total response time is the sum of the execution time of the application (schedule length or makespan) and its in-queue waiting time, if the scheduling platform is shared with other applications. We are interested mainly in minimizing the expected value of the response time, given a set of randomly distributed input values (activity times). Accordingly, makespan and waiting time minimization policies are of main interest here and are discussed in more detail. In particular, we outline relevant parts of Graham and Allahverdi’s taxonomy of parallel processing scheduling problems and portions of Kendall’s queuing theory notation, which will be employed jointly throughout this thesis to describe the problem settings of interest. Both classifications are extended to cover problems with a conditional and iterative control flow in the application model, i.e., the workflow net model, which will be discussed in detail in Chapter 5.

As it follows from the assumptions formulated above, the scheduling policies of interest here must accept probabilistically distributed inputs but also have likewise distributed outputs, i.e., complete schedules. Moreover, any partial schedule whose length might have to be evaluated to make a decision during the scheduling process is distributed. Hence a summary of the relevant subset of probability theory is provided, followed by an in-depth discussion of the problem of evaluating schedules which are functions of random variables. This problem is \#P-complete even in the context of Program Evaluation and Review Technique (PERT), which does not entail an allocation problem and therefore forms a subset of the scheduling problem classes investigated here. This motivates approaches that can estimate schedule lengths accurately and efficiently, which are one of the contributions of this work.

On the other hand, the problem of minimizing the schedule length with arbitrary precedence constraints and communication delays is \(NP\)-hard even with deterministic inputs. This motivates the adoption of heuristic policies to solve it, especially considering the added complexity of managing uncertainty. In particular, the list scheduling
Chapter 2. Background and Preliminaries

approach is examined. Its basic idea is that jobs be ordered into a priority list and that each job be assigned to the processor that allows for its earliest finish time, starting with the highest priority job. List schedulers are known to be able to efficiently solve both deterministic and stochastic scheduling problems with time-domain objectives such as makespan and weighted completion time. These algorithms are also amenable to modifications, such as replacing deterministic input values with estimates of random variables, as it is the case in stochastic scheduling. In general, different estimation methods of activity runtimes, path length estimation strategies and priority list determination mechanisms can be easily introduced (“plugged in”). We outline the generic priority-based list schedulers referenced in the following chapters, including their lazy, greedy and idle-time charged variants and some modifications of the list scheduling pattern, such as the dynamic level scheduling algorithm. Different task priority attributes, such as bottom and top levels and critical path-based attributes, are described as well.

Given the known complexity of the distributed makespan computation problem, the policies proposed in this thesis are evaluated experimentally with a simplified procedure. It generalizes Kwok and Ahmad’s normalized schedule length method and extends it to the stochastic context. For each single realization of the vector of the application’s activities (input) the application’s critical path is computed. Successively, the value of the objective function for the schedule is determined: in our case, the makespan or the response time. The quotient of these two values is the normalized value of the objective. For the purpose of performance measurement, we consider this quotient as the output of the policy. The values of the quotient for all the realizations of the inputs form thus the performance space of the policy with respect to the given objective and set of inputs. The performance measure for the scheduling policy of interest is assumed to be the expected value of the output (the average value in the performance space). The measure of the robustness is given by the standard deviation across the space, with positive and negative semi-deviations indicated separately. The performance space and the statistical dispersion of the outputs are also characterized in more details visually by means of five-number summaries.
Chapter 3

Offline Scheduling of DAG-reducible Stochastic Workflows

In this chapter the problem of scheduling a single known workflow is investigated, under the assumption that the workflow schema (application model) can be expressed by a directed acyclic graph and that the task computation times and inter-task communication times are given by known, arbitrary probability distributions.

The following contributions are put forward:

1. a novel and efficient policy for the problem of minimization of the expected makespan when a single workflow instance is scheduled to either a homogeneous or uniformly heterogeneous platform, i.e., the $P/Q|v_i, c_{ij} \sim {stoch, prec}|E[C_{max}]$ problem (see [2.12]): the policy is based on an original, configurable trade-off mechanism between the gain, associated with greedy (immediate) scheduling of low priority jobs, and the loss, represented by the stochastic delay that such greedy scheduling may induce for higher priority jobs,

2. a novel method of improving the accuracy of the estimates of the stochastic variables associated with the workflow’s activities, based on the employment of partial moments of the relevant probability distributions,

3. an evaluation of the solution with respect to its robustness (see [2.17]),

under consideration of the comparable state-of-the-art algorithms known from the subject literature at the time of this writing.
3.1 Motivating Scenarios

In many real-life scenarios, described by the following non-exhaustive list, there is a need to provide an allocation of linked activities that is immutable in the life cycle of the workflow. In such scenarios, an allocation is sought that must be known before the execution of any instance of this workflow, does not change during its execution and may be repeated in the same manner for the subsequent instances. In other words, a schedule known \textit{a priori} with fixed control- and data flow paths is either a strict requirement or at least desirable: This schedule must provide the information which tasks are to be executed by what processors (but specific timing for the activities and their sequence are usually not prescribed, thus the term “allocation” is here actually more appropriate than “schedule”). Similar requirements and scenarios can be individuated in very different and seemingly unrelated domains.

A first group of such scenarios are those that by design require a fixed deployment of functional components to a set of processing units: This is the case in application-specific integrated circuits (ASIC), embedded systems and other specialized hardware. Such systems are found, for example, in the automotive domain. A possible task graph of an anti-lock break system (ABS) found in most modern automobiles is shown in Figure 3.1. This example is due to Aleti and Meedeniya [7], who proposed a Bayesian-learning-inspired heuristic for optimizing such deployments with respect to transmission reliability, communication overhead and schedule length (in a deterministic problem setting). Without going too deep into specific technical details, it is sufficient in this context to note that the different components need to communicate with one another according to know patterns (i.e., that form a task graph with communication links). They must be deployed to a set of Electronic Control Units (ECUs) with different speeds, connected through a set of likewise heterogeneous buses. This deployment (allocation) and the sequence of actuation is fixed, i.e., “imprinted” in the hardware and immutable, even though the actual execution and communication times are usually allowed to fluctuate.

Applications with more freely deployable components may present similar challenges. The scenario in Figures 3.2 and 3.3 illustrates a cloud application that includes a front-end server (responsible for the application’s presentation, e.g., a HTTP server) and, in the back-end, an application server that appropriately forwards the client’s requests.
3.1. Motivating Scenarios

Figure 3.1: Task graph of the anti-lock breaking system (ABS, following [7]).

to a specialized heavy-weight computing engine (HCE), based on data retrieved from secure web services from a third party and on a large amount of data stored in a pool of databases accessed through a cache. It would be theoretically possible to deploy each task in this application to a potentially large number of physical machines or virtual private servers, such as Amazon Elastic Computing Cloud instances or comparable competitors (in fact, such approaches are common in load balancing implementations). Then an online scheduler could, theoretically, decide at runtime which machine should execute which task for a given request. However, our scenario presents some peculiarities. Whereas HTTP servers and application servers can be deployed freely and used immediately, the employed computing engine (HCE) requires a license and therefore cannot be deployed on many or moved freely between virtual machines at runtime. Moreover, it checks other prerequisites and fetches data from a registered secure third party set of web services, before processing a request: in practice, it would incur an unacceptable start-up penalty if “unprepared”. Additionally, the databases contain extremely large amounts of data and for the caching mechanism to work efficiently, the cache has to be refreshed periodically and filled at some time before the request is processed. Even if real-time monitoring is in place, an online scheduler cannot fulfill such requirements.

In a more abstract view, in the above scenario the key difficulty that precluded the use
Figure 3.2: Functional view of the cloud application scenario.

Figure 3.3: A possible workflow schema of the cloud application scenario from Figure 3.2.
3.1. Motivating Scenarios

Let us consider the scenario in Figure 3.4 showing a simplified skilled migrant visa processing workflow. When executing such a workflow, the office manager must take into consideration additional requirements. For example, an employee in charge of processing sensitive personal data undergo a specific training and certification and that channels where confidential data are transmitted be appropriately secured (hence, they may not be allocated *ad-hoc*). This corresponds to the requirement of a task-specific configuration phase for at least a subset of the processors and channels, as in the previous example, and therefore, precludes the employment of online scheduling in a similar way.

A second large group of such scenarios are those in which real-time execution monitoring is not possible, because of disproportionately high management costs. Finally, in many cases there is simply no proper “scheduler”, i.e., no entity that can dispatch tasks in real time. In others, runtime instructions (online scheduling) or corrections (rescheduling) are not desirable, as this would induce delays due to communication with the central scheduler or a computation overhead that would be significant if compared to the total processing time of the workflow.

All of these scenarios pose requirements that preclude the application of online policies, as postulated, e.g., in stochastic scheduling. At the same time, the assumption of
deterministic task processing times and communication delays, which is the basis of most of the heuristic approaches developed in the area of parallel processing scheduling, does not hold in these and other realistic applications.

### 3.2 Problem Formulation

Let $G = (J, E)$, with $J$ the set of nodes and $E$ the set of edges connecting them, be a single directed acyclic graph (DAG) representing a process model, or workflow schema. The nodes represent atomic tasks and the edges represent precedence constraints between them. Each task has an associated execution time and each edge between two tasks has an associated communication delay. Each task has an associated nominal execution time (the required processing time at normal speed) and each edge between two tasks has an associated time, that signifies the nominal communication delay that scheduling those two tasks to two different processors incurs. In other words, we follow the classical assumption that a delay is only caused when an actual transfer of data between two separate processing units is required, depending on the scheduling decisions taken (see 2.2.2), in a manner proportional to the amount of information to be transferred.

In addition, let all the execution times $w(j_i)$ and communication times $w(e_{ij})$ be described by random variables $w(\cdot)$, with arbitrary probability distributions, as outlined in Sect. 2.2.1. In particular, it is irrelevant whether the distributions are part of any specific family and whether they are continuous or discrete. The sole restriction is that their first and second partial moments be defined and finite (see 3.5.1).

Let $Q$ with $|Q| < \infty$ be a set of parallel processors and $Q \times Q$ a set of communication channels connecting them.

Both the case of identical and uniform processors and channels, i.e., both a homogeneous and a uniformly heterogeneous environment are assumed in the following. Thus, the effective execution time of a job is given by its intrinsic weight (see 2.1) multiplied by the speed of the processor the job is assigned to. The same principle applies to the communication channels.

The atomic tasks of the workflow are allocated to idle time slots of these processors and executed without preemption: a task, once its execution has commenced on a
The $|\text{prec}|E[C_{\text{max}}]$ minimization problem under the outlined assumptions is formulated by referring to the following definition.

**Definition 3.1.** Let $P$ be a $|\text{prec}|C_{\text{max}}$ problem and $S = (\{S_j\}, \{C_j\}, \{p(j)\})$ a solution to $P$. An order-invariant transform of $S$ is a tuple $S'((\{S'_j\}, \{C'_j\}, \{p'(j)\}))$ such that:

- $\forall j, k \in V : p(j) = p(k), S_k \geq C_j \Rightarrow p'(j) = p'(k), S'_k \geq C'_j$
- $\forall j, k \in V : p(j) \neq p(k), (j \prec k) \in E \Rightarrow S'_k \geq C'_j + w(e_{jk})$

In other words, the order-invariant transform $S'$ of a schedule $S$ is another solution to $P$ which retains the allocation of all tasks to the processors and the sequence on every processor and may be different from $S$ in that it assigns different, but feasible, start and finish times to the tasks. Note that for a single $S$ there are infinitely many such transforms. They are interesting in the light of the fact that, in practice, a schedule $S$ is always computed for a realization vector of the processing and communication times and in theory it also corresponds to at least one such realization (e.g., the one based on expected activity run-times). The $|\text{prec}|E[C_{\text{max}}]$ minimization problem can now be formulated as follows: find a schedule for which the set of order-invariant transforms generated for a representative set of realization vectors is such that the average makespan of a transform is minimized. Note that this formulation provides practical means of evaluating a solution $S$ despite the #P-completeness of the distributed makespan evaluation problem (Sect. 2.13).

### 3.3 Related Work

Surprisingly few attempts have been made to tackle this particular problem. One of the reasons is the general reliance on online scheduling algorithms in most settings characterized by uncertainty, because of their proven superiority with respect to their offline counterparts. For example, a study by Lawrence and Sewell [146] presents extensive experimentation to show that relatively simple online algorithms schedule
both independent jobs and task graphs more optimally and efficiently than the most sophisticated offline algorithms.

The authors point out the following. Firstly, it is not trivial to apply an optimal solution technique for a simplification or abstraction of a scheduling problem to the original, more complex problem: e.g., a solution to a stochastic scheduling problem obtained by reducing it to a deterministic one is of limited value for the original problem. Secondly, no matter how elaborate an offline scheduling algorithm might be and how extensive the information extracted from the known probability distributions is, the solutions given by offline algorithms will quickly deteriorate as the uncertainty grows. In other words, a simple sensible online scheduling policy will yield better results in terms of a makespan objective than the most elaborate static algorithm forced to work with uncertain information.

Online algorithms can admittedly handle even non-stochastic uncertainty, i.e., one that is not quantifiable in terms of probability distributions. Therefore, they can work with non-clairvoyant scheduling models (see 2.13). These, however, are not of central interest here for the presented research: it focuses on the better-behaved stochastic uncertainty model. In the area of stochastic scheduling that has traditionally investigated solutions for these models, results are expressed in the form of optimal or near-optimal online policies for a set of interesting objective functions, formulated as expectations on random variables such as makespan [56] or weighted completion time of a set of jobs [201].

In fact, it is often argued by prominent authors in this field (e.g., in the works of Pinedo [188] and of Möhring et al. [173], Uetz [236] and Skutella and Uetz [221]) that in stochastic settings, the problem naturally reduces to finding policies for online scheduling decisions, and should not be treated as a combinatorial optimization problem to be solved offline. For a discussion on this issue, refer to Sect. 4.3. As illustrated in Sect. 3.1, however, an online approach is in fact not always applicable or desirable. Moreover, none of the stochastic scheduling policies known from the subject literature is designed for application models with optional communication delays (see 2.12). These extend in fact the notion of ordinary precedence constraints and their presence make the scheduling problem strongly NP-hard.

A genetic algorithm for static scheduling in a similar setting was reported by Yang et al. [270]. However, it only supports distributed task durations, not communication
3.3. Related Work

times. Tang et al. [227] proposed a directly related scheduling algorithm, the stochastic heterogeneous earliest finish time heuristic (SHEFT), based on HEFT [233, 234]. SHEFT is the sole list scheduling heuristic applicable in the very same problem setting \( (Q|v_i, c_{ij} \sim \text{stoch}, prec|E(C_{\text{max}}) \text{, see Sect. 2.12}) \) known from literature at the time of this writing, although only for exponentially distributed times. Additionally, Tang et al. applied known static scheduling algorithms, such as HEFT and DCP (Dynamic Critical Path [140]) in the same setting, “porting” them to handle a stochastic setting by using distribution means as activities’ durations. SHEFT replaces the mean activity durations by the sum of mean and standard deviation of the corresponding distributions, when computing the task priorities and the feasibility of the insertion of a lower priority task into an idle time slot existing in the previously generated partial schedule. The authors prove experimentally that SHEFT performs better than the reference mean-based HEFT and DCP algorithms and that HEFT outperforms DCP in the given setting.

The baseline algorithm on which HEFT and SHEFT, among others, are based is shown in Alg. 7 below. For an insertion to be feasible, the scheduling model must be offline and the previously computed partial schedule must contain sufficiently large idle time slots. This means that the idle time slot must be greater than or equal to the processing time of the task considered for insertion, hence the term “safe-insertion”. The processing time is either known (in a deterministic context) or estimated by the algorithm’s logic. However computed, this estimate is static. When adopting the same task priority determination mechanism, HEFT and SHEFT only differ with respect to the scalar weights assigned to the activities in the DAG.

Algorithm 7: A generic offline safe-insertion list scheduling algorithm for the makespan or expected makespan objective.

| input: | an instance of \( Q|prec, c_{ij}|C_{\text{max}} \) or \( Q|prec, c_{ij}|E[C_{\text{max}}]; \) a job priority list \( L \) |
| output: | a feasible schedule |
| while \( L \) has unscheduled jobs do |
| \( j \leftarrow \) first job in \( L \) \( \triangleright \) the highest priority job in the list |
| \( p \leftarrow \) the processor that minimizes the estimated finish time of \( j \) with insertion into idle times if constraints are respected and other tasks need not be shifted |
| schedule \( j \) to \( p \) |
| remove \( j \) from the list \( L \) |

This algorithm is still greedy in the sense that the outcome schedule does not always reflect the priority list, as in lazy algorithms, which are the only ones to guarantee that a lower priority task is never scheduled before a higher priority one.
3.4 Problems with Existing Solutions

The studies reported in this chapter individuate four distinct issues regarding the existing algorithms. Two are related to the adopted baseline algorithm (HEFT): on the one hand, HEFT implies the use of average processor and communication channel speeds for the job priority determination; on the other, the exploitation of idle time slots is carried out on an is-feasible basis without additional checks.

In addition, the assumption that all the activities be exponentially (or normally) distributed is not realistic in many applications.

The final issue in the list is related to the accuracy of the estimates of the stochastic activity durations used as the input to the job prioritization method (whichever method is employed).

Zhao and Sakellariou [274] provide experimental evidence, using four metrics, that in a deterministic setting, HEFT tends to perform better when upward rankings (t-levels, see Sect. 2.16.1) are used and, more importantly, when these are computed using best- or worst-case task run times, not average values. Note that average, median, best and worst values are due to the heterogeneity of the processors in the considered $Q_{\text{prec}}C_{\text{max}}$ setting, not to the computation times being assumed as random variables. Moreover, the treatment of communication times is not explicitly investigated in the study. However, the study still does point out an important shortcoming: the reliance on the average speeds of the processors and communication channels and the employment of average speeds in the priority list computation procedure. Empirical evidence seems to suggest that these values are not accurate, yet the cited study does not provide an explanation. It should be noted that the tasks considered for scheduling are always assigned to the fastest available processor when an EFT heuristic is employed (including, naturally, in HEFT), thus the average processing time for such tasks almost never corresponds to the actual one. This shortcoming is slightly mitigated by the fact that, during scheduling, some (typically lower priority) tasks will in fact be scheduled to slower processors. Most higher priority tasks, however, are completed faster than their average processing time specifies. On the other hand, the EFT mechanism used for processor selection, while taking into account the speed of the selected processor, ignores the speeds of its outgoing channels. Therefore, it makes sense to assume that the communication delays on the given set of physical channels are approximated by
3.4. Problems with Existing Solutions

the corresponding average delays (as it is done in HEFT), as the channels are not selected on the fastest available basis. Overall, in a uniform computing environment, the processing times, especially for higher priority tasks, tend to be overestimated with respect to the communication delays. This results in an inaccurate estimation of priorities, which has been traditionally propagated to all HEFT-inspired algorithms.

The second problem with a HEFT-like algorithm, when ported to a stochastic setting, is illustrated as follows. The insertion in Alg. [7] is always safe only in the deterministic context: with stochastic activity durations the actual idle time slot length in a schedule realization may be smaller than statically estimated, e.g., based on mean activity durations. Thus an inserted lower priority job may delay a higher priority one in particular schedule realizations and no provision is made to prevent this effect or estimate its impact. Assuming that the employed prioritizing mechanism is optimal in the majority of the cases, a consistent violation of the priorities without any safeguard to limit its effect may be detrimental for the outcome schedules.

In the introduction to this section, the problem of too restrictive and unnatural assumptions with respect to the probability distribution families admitted has also been mentioned. Many simpler stochastic scheduling problems can in fact be solved optimally when the jobs’ execution times are assumed to have exponential distributions (but there must be no communication delays). This is prominently the case in stochastic scheduling. Similarly, e.g., in queuing theory many closed-form results can be derived in analogous cases, i.e., for “well-behaved” Markovian queues (see Sect. [2.18]). The reason is the particular mathematical amenability of these distributions. In realistic scenarios, such as those previously presented in this chapter, such an assumption is very hard to justify. For example, the whole body of work in Program Evaluation and Review Technique (PERT, see [2.15.1]), concerning the computation or estimation of distributed makespans in project scheduling is motivated by the fact that realistic projects seldom have such well-behaved activities. The adoption of the sum of mean and standard deviation as an approximation of the actual time in Tang et al. [227] is justified for SHEFT on the basis of a theorem proved by Skutella and Uetz [221]. Under the assumption of exponentially distributed activities the normalized standard deviation is exactly one for all jobs and the sum of mean and standard deviation may indeed be a valid estimate for problems covered by the theorem.

This result, however, is proved only for online scheduling policies applied, in a ho-
mogeneous processing environment, to precedence-constrained job sets without communication delays and when the objective is to minimize the (weighted) sum of job completion times. There are reasons to conjecture that this result could be extended to objective functions that, like the weighted sum above, are both regular and additive (see Uetz [237]). However, there is no proof that the result can be carried over to other than homogeneous processing environments and, especially, problems with communication delays, which is tacitly assumed in Tang et al. [227].

The final issue with the existing solutions regards the choice of estimates that are used in non-trivial problem instances for tractability reasons to replace the full probability distributions of the stochastic variables of interest.

All list scheduling approaches perform essentially two operations for the purpose of job priority determination: sum of durations for activities being in a sequence and computation of the maximum of durations for activities (or activity sequences) when, before the execution of some job, the paths including its parent jobs are synchronized (joint). As it is known (see 2.15 and 2.15.1), the exact computation of both is a \#P-complete problem, so, given the necessity of performing a large amount of such operations repeatedly, easily computable, comparable and manageable estimates are needed. This suggests single real-valued numbers (scalars) rather than, e.g., formulations using vectors. There exist statistical analysis methods, such as five-number summaries, which are good means of summarizing the degree and direction of dispersion of a distribution. In fact, they are also used for the presentation of experimental results in this thesis, but would not be easily manageable in a large amount of operations as outlined above. Additionally, for the sum operator, actually the expected values rather than medians are good estimates, as they are the most likely values of the stochastic variables of interest. In fact, the expected value of the sum is the sum of expected values even if the variables are not independent. The issue arises with the non-linear maximum-of (\(max\)) operator used on joins. Neither the expected value nor the standard deviation of the maximum can be as trivially derived from the operands as in the case of the sum. Moreover, the very specific meaning of the \(max\) operator makes it actually meaningless to use non-directional dispersion measures in the presence of generally asymmetric distributions. In particular, as noted in Sect. 2.9, the standard deviation, while commonly used in probability calculus as a measure of statistical dispersion, is one such non-directional measure, i.e., it does not contain any information on which tail of a univariate distri-
3.5. Solution Approach

In the following, this chapter provides an approach, structured by sub-problems coinciding with the issues outlined in the previous section.

The following subsection is dedicated to outlining a novel method for the determination of accurate scalar estimates for the static handling of stochastic variables. This is done while additionally lifting any significant restrictions on the nature and shape of the probability distributions of the activities, i.e., addressing issues three and four in Sect. 3.4. Subsect. 3.5.2 addresses the remaining ones, by proposing an original approach for priority list refinement in stochastic scheduling settings, based on a trade-off mechanism.

The proposed improved estimates can be plugged in and used in any job priority determination mechanism, whereas the trade-off based refinement is likewise applicable to any priority list. Therefore, in Sect. 3.6 both proposed improvements are evaluated when integrated with state-of-the-art several prioritizing heuristics.

3.5.1 Partial Moments in Path Length Estimation

In Sect. 2.9 the concepts of mean, variance and standard deviation, universally used in probability calculus and in scheduling in stochastic problem settings were presented.

The set of definitions and properties which follows in this section is due to Winkler et al. [263]. It is a parametric formalization of the concepts of mean, variance and higher moments, where the parameters define an interval over which the respective moments are derived. Such moments are termed partial or incomplete moments (partial expectation, variance etc.) as opposed to the respective (complete) moments of Sect. 2.9.
Chapter 3. Offline Scheduling of DAG-reducible Stochastic Workflows

This formalism follows early works by Pearson and Lee [184], who defined the incomplete moments for standard normal functions, Gurland [95], who defined incomplete moments for the general univariate case, Kamat [124, 125] and Nabeya [178] for the multivariate normal distribution, and others. Shafer [213] offers a broader perspective on the matter from a modern point of view.

Winkler et al. [263] suggest that complete moments should be replaced by partial ones whenever only a subset of the values of a random variable is of interest. In fact, the partial moments have a long history of use ([263]) in many diverse domains: Bayesian point estimation (Raiffa and Schlaifer [214]), inventory theory (Arrow et al. [15], [14]), Veinott [253], theory of the firm and production (Roodman [200], Baron [24], Horowitz [108]), asset divestiture and stopping rules (Hayes [100], DeGroot [60]), screening and classification with loss functions (Marshall and Olkin [166]). A very straightforward example comes from the domain of financial risk analysis. In principle, a large standard deviation reflects a large uncertainty, which may have adverse effects. However, if a probability distribution models a cash flow, a large standard deviation is not necessarily unfavorable, as long as it is mainly contributed to by a large positive semi-deviation, i.e., the distribution has a large positive tail. This means that larger than expected income values are more likely than smaller than expected ones, which is also usually positive in the common sense of the word.

The complete first moments, on the other hand, are obviously commonly considered in scheduling problems with stochastic uncertainty (see previous section), especially when well-established deterministic scheduling approaches are adapted to handle such stochastic settings.

However, this thesis, as far as the author is aware, is the first instance of the use of partial moments (and, in subsequent chapters, also conditional and conditional partial moments) in the design of a heuristic for workflow scheduling problems. Unlike in Winkler’s cited work, the partial and conditional partial moments are defined on intervals parametrized on both bounds. The rationale for this is that particular, bounded regions of the probability distribution are investigated depending on the context (see later part of this section).

Definition 2.23 gives the mean (expectation) of a random variable $X$ over the whole interval, on which the density function is defined. This is the complete expectation of $X$ and corresponds to its marginal mean. The following definition is the interval-based
3.5. Solution Approach

Definition 3.2. The partial expectation of a random variable $X$ over an interval $[y, z]$ is given by:

$$E_y^z[X] = \int_y^z xf(x)dx = \int_y^z xF(x)$$  \hspace{1em} (3.1)

Definition 3.3. The $n$-th partial moment of a random variable $X$ about the origin, over an interval $[y, z]$, is the expectation of $X^n$ over $[y, z]$, written: $E_y^z[X^n]$.

Definition 3.4. The partial moment generating function of a random variable $X$ about the origin, over an interval $[y, z]$, is given by:

$$M_y^z(t) = E_y^z[e^{tX}] = \int_y^z e^{tx}dF(x)$$  \hspace{1em} (3.2)

The $n$-th partial moment of a random variable $X$ about the origin, over an interval $[y, z]$, is given by:

$$E_y^z[X^n] = \int_y^z x^n f(x)dx = \int_y^z x^n dF(x) = \frac{d^n}{dt^n}M_y^z(t)|_{t=0}$$  \hspace{1em} (3.3)

Note that in Winkler [263] only the partial expectation is given with a parametrized lower bound ($y$), while the $n$-th partial moment is defined over the semi-open interval $(-\infty, z]$ for all $n > 1$.

The partial moments become complete moments for $y \to -\infty$, $z \to \infty$:

$$E[X^n] = \lim_{y \to -\infty, z \to \infty} E_y^z[X^n]$$  \hspace{1em} (3.4)

and, conversely:

$$M(t) = \lim_{y \to -\infty, z \to \infty} M_y^z(t)$$  \hspace{1em} (3.5)

The $n$-th partial moment of a random variable $X$ about a point $x_0$, over the interval $[y, z]$, is given by:

$$E_y^z[(X - x_0)^n] = \sum_{i=0}^{n} \binom{n}{i} (-x_0)^i E_y^z[X^{n-i}]$$  \hspace{1em} (3.6)
Chapter 3. Offline Scheduling of DAG-reducible Stochastic Workflows

The novel idea here is to provide a refinement for the estimate provided to the \( \max \) operator, by using a relatively simple measure of dispersion pointing in the same direction as the operator (towards infinity). This measure is the part of the standard deviation that is contributed to only by the values that are larger than the mean \( \mu \), i.e., the values \( x_k \) that satisfy: \( \max(x_k, \mu) = x_k \).

If we substitute the marginal mean \( \mu \) for \( x_0 \) in Eq. 3.6, we obtain the definition for the \( n \)-th partial moment about the mean over the corresponding interval. Obviously, only the second and upper moments can have a non-zero value. In the following, we are only interested in the second partial moment (variance) and, more specifically, the derived partial standard deviation, as stated above.

**Definition 3.5.** The positive semi-variance of a random variable \( X \), written \( \text{Var}^+ [X] \) or \( \sigma^2_+ \), is:

\[
\text{Var}^+ [X] = \sigma^2_+ = \int_{\mu}^{\infty} (x - \mu)^2 dF(x)
\]  

(3.7)

The negative semi-variance of a random variable \( X \), written \( \text{Var}^- [X] \) or \( \sigma^2_- \) is defined analogously and it holds that:

\[
\text{Var} [X] = \sigma^2 = \text{Var}^+ [X] + \text{Var}^- [X]
\]  

(3.8)

The positive semi-deviation \( \sigma_+ \) is the square root of the positive semi-variance, i.e., it is defined analogously to the total standard deviation.

The negative semi-moments are defined analogously, but are of less importance for the application to the scheduling problem. However, it is fairly easy to propose a problem where these could be the estimates of choice: namely, any problem that is analogous to the problem at hand but uses the \( \min \) instead of the \( \max \) operator, e.g., when applying, to the corresponding stochastic variables, the aggregation function for reliability or availability as in, e.g., Zeng et al. [273].

### 3.5.2 The Stochastic Trade-off Insertion Mechanism

Sect. 3.4 outlines the issues related to the adoption of an algorithm such as HEFT as a baseline solution for the problem at hand. The adoption of average processor and
3.5. Solution Approach

Channel speeds to compute weighted attributes (t-levels or similar) is not justified either theoretically or experimentally, therefore average speeds are replaced by nominal (assignment-independent) speeds in the proposed approach. As such, the method of determining the priority list is external to the second problem outlined in the above mentioned section. This problem is not specific to HEFT, but to any safely greedy algorithm that inserts low priority tasks into idle time slots in a partial schedule, based on deterministic values of the variables of interest. In a setting where such variables are in fact stochastic, the predictions on the safe use of such slots become also uncertain.

The approach proposed to solve this sub-problem is based on the idea of stochastic trade-offs. The idea of a trade-off-based mechanism is motivated by the properties of the makespan objective that are particularly relevant in a stochastic setting. The minimization of the global makespan is an NP-complete problem even for deterministic DAGs (Sect. 2.14): The global objective can only be optimized by exploring the whole state space. List scheduling heuristic approaches choose to solve a set of related, tractable problems instead: First, a job priority list is determined, then, in a step-wise fashion, the job of highest priority which is still unscheduled is chosen and, for this job, the earliest finish time is minimized, across all possible job-to-processor allocations. In other words, the globally intractable makespan problem is transformed into a sequence of locally defined, polynomial problems of minimizing the earliest, or expected earliest finish time (EFT) of the job that, at each step, has the highest priority.

In the general case, this results in leaving (wasted) idle time on the processors, as tasks of higher priority that are not ready for execution may cause the scheduler to wait. On the other hand, schedules with less interleaved idle time are obviously closer to the optimum. Strictly speaking, it is the idle time that could be used to process a job that is ready for execution that is wasted. This observation suggests to simultaneously minimize a second local objective: the amount of wasted idle time. In the “classical” deterministic setting, assuming that the priority list is accurate, the only sensible strategy is to insert a lower priority task into an idle time slot, if that slot is not smaller than the task’s execution time on that processor: Otherwise it is guaranteed that a higher priority task will be delayed. This is actually done by any safely greedy algorithm (see 2.16.2 Alg. 3).

In the setting considered here, however, the activity execution times are stochastic. Therefore, it is possible to calculate the probability that an idle time slot will be large
Chapter 3. Offline Scheduling of DAG-reducible Stochastic Workflows

enough for an insertion: This, however, is a \#P-complete problem in its own right. Alternatively, it is possible to compute the probability distribution of the delay that may be caused to the finish time of a task of higher priority by the insertion a task of lower priority. This distribution represents a potential loss. The distribution of the gain in terms of the amount of idle time recovered by the insertion can be determined similarly. The trade-off mechanism is designed to balance these two local objectives. Effectively, the optimization problem of a single global objective function such as the expected makespan is mapped to a problem of locally minimizing two objective functions: the finish time of the task considered for scheduling at a given step and the maximum delay that this decision may cause in the finish time of any task of higher priority. For a general idea of the approach, see listing 8.

Algorithm 8: A generic formulation of TOFF/\(\epsilon\): the proposed trade-off-based policy, driven by a threshold parameter \(\epsilon\).

1  input: an instance of \(Q|\text{prec, } v_j, c_{ij} \sim \text{stoch}|E[C_{\text{max}}]\); a threshold value \(\epsilon\)
2  output: a feasible schedule
3  while there are unscheduled jobs do
4     compute a job priority list \(L\) provision for a dynamically computed priority list
5     consider the highest priority job \(j\) still in \(L\)
6     foreach processor \(p\) do
7        estimate the finish time of \(j\) with no insertion
8        estimate the finish time of \(j\) with insertion into the earliest idle time slot, if feasible
9        estimate the gain \(\beta\) achieved by the insertion
10       estimate the resulting delay \(\beta'\) for any jobs already scheduled
11       compute trade-off values \(\theta\) for the gain and loss estimates
12       if any of the trade-offs is above the threshold \(\epsilon\), consider the next idle time slot
13       record the best completion estimate of \(j\) with an acceptable trade-off value \(\epsilon\) this may be the no-insertion estimate
14       schedule \(j\) to the processor \(p\) that minimizes the finish time estimate of \(j\) with an acceptable trade-off
15       remove \(j\) from the job list

As per Sect. 2.15 the problem of computing such distributions is also \#P-complete in the general case. Therefore a computationally efficient mechanism is needed to perform a similar function. Moreover, what can be considered an acceptable trade-off should depend on the relative priorities of the tasks involved. The policy is designed in a way that allows for a parametric specification of the trade-off threshold (\(\epsilon\)). Algorithm 9 outlines the proposed approach in more detail. The policy variable \(\pi\) is used as a means of parametrization of the adopted priority list determination mechanism (Lines 3, 8, 16), e.g., t-level-based, and of the adopted strategy of estimating randomly dis-
tributed activity durations, e.g., based on the mean values or on the sums of mean and positive standard deviations. The latter is employed by the function \textit{EstimateTime}, called in Lines 6, 10, 17, which trivially returns the estimated start and finish times for the currently considered job on the given processor, according to the adopted estimation strategy and the feasibility conditions as per Def. \ref{def:estimation}. The estimates based on a no-insertion approach are cached (Line 6) and compared with those based on insertion (Lines 9-10), starting with the earliest idle time slot on the given processor for which the insertion is feasible (Line 7). If there is no gain from the insertion, the remaining computation is skipped for the current processor (Lines 12-13). Otherwise, a trade-off $\theta$ is computed (Line 20) between the locally defined loss (the estimated delay in the completion of a higher priority job) and gain (the estimated time the insertion can speed up the completion of the current job). The original priorities $l, l'$ act as a discount factors for the gain and loss (Lines 11, 19) in the determination of the trade-off value $\theta$ (Line 20). The trade-off $\theta$ thus expresses a measure of the value of the time gained (including recovered idle time) against the value of the time potentially lost. It is compared to a threshold value $\epsilon$ (Line 21) which is one of the inputs of the policy. If any of the generated trade-off values exceed the threshold, another idle time slot is considered (Line 22): failing that, the job is scheduled with no insertion (lazily). A successful insertion additionally shifts the impacted previously scheduled jobs (Line 31). Note that this only affects \textit{estimated} start and finish times and can be delayed until the schedule is recomputed for a realization vector of the activity durations. Therefore, it does not impact the complexity of the algorithm, which is discussed in more detail in the next section. It should also be pointed out that this design of the trade-off mechanism reduces the theoretical computational complexity by employing scalar estimates
instead of probability distribution functions.

Algorithm 9: TOFF(\(\pi\), \(\epsilon\), \(S\))

1. \(U \leftarrow Unscheduled.Jobs(S)\)
2. while \(U\) not empty do
   3. \(j \leftarrow Priority.Job(\pi, U)\) \(\triangleright \) consider the next job in the list
   4. \(M \leftarrow \{\}\) \(\triangleright\) temporary set of potential mappings for \(j\)
   5. foreach processor \(p \in Q\) do
      6. \((ST, FT) \leftarrow Estimate.Time(\pi, j, p, S, M)\) \(\triangleright\) the earliest start/finish estimates of \(j\)
         with no insertion
      7. foreach idle time slot \((STI, FTT)\) on \(p\) in increasing order of \(STI\) do
         8. \(l \leftarrow Priority(\pi, j)\)
         9. schedule \(j\) at \(STI\) in the in a copy of the state \(S^*\)
         10. \((ST^*, FT^*) \leftarrow Estimate.Time(\pi, j, p, S^*, M)\)
         11. \(\beta \leftarrow l'(FT^* - FT)\)
         12. if \(\beta = 0\) then
             13. break loop and consider another processor
         14. \(\beta' \leftarrow 0\)
         15. foreach previous job mapping \(m' = (j', p', ST', FT') \in M : ST' > ST^*\) do
            16. \(l' \leftarrow Priority(\pi, j')\)
            17. \((ST'^*, FT'^*) \leftarrow Estimate.Time(\pi, j', p', S^*, M)\)
            18. if \(l'(FT'^* - FT') > \beta'\) then
                19. \(\beta' \leftarrow l'(FT'^* - FT')\)
            20. \(\theta \leftarrow \frac{\beta'}{\beta}\)
            21. if \(\theta > \epsilon(\pi)\) then
                 22. break loop and consider another time slot \((STI, FTT)\) on \(p\)
            23. \((ST, FT) \leftarrow (ST^*, FT^*)\)
            24. \(M \leftarrow M \cup \{(j, p, ST, FT)\}\) \(\triangleright\) add a new potential mapping for \(j\) to \(M\)
   25. \(FT_{min} \leftarrow \infty\)
   26. \(m_{min} \leftarrow \text{null}\)
   27. foreach mapping \(m = (j, p, ST, FT) \in M\) do
      28. if \(FT < FT_{min}\) then
         29. \(m_{min} \leftarrow m\)
      30. schedule \(j\) according to \(m_{min}\) \(\triangleright\) saves \(m_{min}\) in the system state \(S\)
      31. shift any jobs as required for the insertion
   32. \(U \leftarrow U \setminus \{j\}\) \(\triangleright\) remove \(j\) from \(U\)
3.5.3 Complexity Analysis

In this section, the time complexity of the proposed scheduling policy is evaluated: the exact complexity ($\Theta$) is provided for the steps for which it is available, the worst-case ($\hat{O}$) complexity for the remaining cases. Note that the standard big-O ($O$) notation is not adopted here as it is not suitable for multiple variables in general (see Howell [109]).

In Sect. 3.5.1 the employment of two partial moments (mean and positive semivariance) is proposed, both for the determination of the job priority list and the estimation of the start and finish times of the jobs during the scheduling process. The estimates can be computed offline, once (at the beginning) for each activity in the application and re-used.

The complexity of the computation of each partial moment of a single probability distribution is linear in the number $s$ of the support points in the discrete representation of its CDF. Thus, the overall exact (not the worst-case or average) complexity of this step is:

$$\hat{\Theta} = 2sa = 2s(v + e)$$

where $a = v + e$ is the total number of activities, $v$ the number of jobs and $e$ the number of the communication edges (precedence constraints). A natural assumption here (which could be trivially lifted, if necessary) is that the number of the support points in the discrete representation of the CDF is the same ($s$) for all activities.

The trade-off-based list scheduling policy shown in Alg. 9 performs $v$ steps to schedule $v$ jobs, as a typical elementary list scheduling algorithm does. In each step, for each of the $p$ processors, first, a mapping without insertion is determined. Next, for at most the number of idle time slots created on each processor by the previous scheduling decisions (there may be at most as many such slots as jobs), a mapping with insertion is determined, one per job and processor, and the trade-off computed for each of the previously effectively scheduled mappings (at most $v$). For each considered job and processor pair, a single mapping is chosen, either with insertion or without. The best (according to the estimated EFT) mapping is chosen across all job-processor pairs. Thus the overall combined, worst-case complexity of the just described steps is given
Chapter 3. Offline Scheduling of DAG-reducible Stochastic Workflows

by:

\[ \hat{O}_{TOFF} = v[p(1 + v^2) + p] = pv(2 + v^2) \]  

(3.10)

The total complexity of any list scheduling policy also depends on the complexity of the algorithm determining the priority list, which is external to the proposed approaches related to the estimators and the trade-off mechanism. For example, the b- and t-level-based prioritizing has a complexity of \( O(v^2) \), while the dominant sequence has one of \( O(v^3) \) (in big-O terms).

Thus the total worst-case complexity of the TOFF/\( \epsilon \)-policy with partial moment-based estimators and a dominant sequence priority list is given by:

\[ \hat{O} = v^3 + vp(2 + v^2) + 2s(v + e) = (p + 1)v^3 + 2(pv + sv + se) \]  

(3.11)

3.6 Experimental Evaluation

For evaluation purposes, the procedure outlined in Sect. 2.19 is employed. A set of 500 graphs is created and, for each single graph from the above set, 100 realizations, totalling 50000 different workflow instances. Each graph has an arbitrary structure, randomly generated by a procedure adapted from the benchmark basis originally proposed by Kwok and Ahmad [142], with additions due to Topcuoglu, Hariri and Wu [233], and parametrized as follows:

- the number of jobs (graph nodes) \( v = |J| \in \mathbb{N} \) is chosen in a uniform random manner from the interval \([10, 25]\),

- given a structural parameter \( \alpha \), such that by definition \( \alpha \sqrt{v} \) is the mean graph height, \( \alpha \) is uniformly distributed in the interval \((0, \alpha_{\text{max}}]\) and the values of \( \alpha_{\text{max}} \) are chosen from the set \( \{0.1, 0.25, 1, 4, 10\} \),

- the maximum out-degree (number of children) of a node \( \max_{j \in J} |J_{ch}(j)| \) is set to 10 and the actual out-degrees of the jobs are uniformly distributed in the interval \([1, \max_{j \in J} |J_{ch}(j)|]\),
3.6. Experimental Evaluation

- the maximum level jump (difference in levels in the graph between communicating tasks) is set to 10 and the actual jumps are uniformly distributed in [1, 10],

- the mean computation times $E[w(j_i)]$ for the jobs are sampled uniformly randomly from [1, 10],

- the computation-to-communication ratio (CCR) assumes values from the set \{0.25, 0.5, 1, 2, 4\}.

Note that only the mean job execution times are taken from a normal distribution with the given support. They are successively used to extrapolate completely arbitrary continuous four-parameter Beta distributions (see Appendix A) such that each distribution has a mean $\mu = E[w(j_i)]$, is supported in $[\frac{\mu}{4}, 4\mu]$ and has a normalized standard deviation $\frac{\sigma}{\mu}$ of a value from the set \{0.01, 0.25, 0.5, 0.75, 1, 1.25\}. The communication delays are likewise assigned four-parameter Beta distributions extrapolated in a manner such that the requirement given by the CCR factor is satisfied. This is a fairly computationally expensive set of steps, which, however, is performed in a pre-simulation phase.

Note that prescribing the value set of the normalized standard deviations implies also their finite values, which is a requirement for both TOFF/$\epsilon$ and, e.g., SHEFT. The adoption of bounded intervals, on the other hand, reflects the observation that in realistic settings any task or communication realization requires at least $\zeta$ time units, where $\zeta > 0$ is an arbitrary constant, and that no task or communication may require infinite time.

Each instance has the same structure and precedence constraints, but different activity execution times which are given by sampling the appropriate distributions.

For any generated task graph with $v$ jobs (nodes), the number of processors $|Q|$ depends on $v$ and a multiplier $p_{mul}$ according to the equation: $|Q| = \lceil p_{mul}v \rceil$, with $p_{mul}$ assuming the values in the range [0.2, 1.0] with a step of 0.1. In the uniform environment, the processor speeds are chosen as uniform random values in the interval (0,1], as are the speeds of all the processor-to-processor physical channels, i.e., the network is fully connected. The speed of any of the processor-to-processor physical channels is defined similarly and independently of the speeds of the connected processors. In the
homogeneous environment, all the speeds are equal to one.

To evaluate the effectiveness of the improved partial moment-based estimation methods and of the trade-off-based policy design proposed in this chapter, the simulations outlined above are run against a number of well-established scheduling policies, grouped by relevant features. Table 3.1 provides more details.

The evaluation is also based on the methodology outlined in Sect. 2.19. For each scheduling policy, the performance metric is the mean of the normalized makespan values achieved under that policy. The normalization is performed, for each workflow instance, by dividing the absolute makespan by the sum of the actual processing times of the jobs in the instance’s critical path. The metric is given by the equation below.

\[
E[C_{\text{max}}(\pi)] = \frac{1}{N} \sum_{i=0}^{N} C_{\text{max}}(i, \pi) = \frac{1}{N} \sum_{i=0}^{N} \frac{C_{\text{max}}(i, \pi)}{\sum_{j \in J_{CP}(i)} r(i, w(j))} = \mu_n(\pi) \quad (3.12)
\]

In addition to the mean, for the normalized makespan, the statistical dispersion of its values is shown of the graphs, given by the appropriate five-number summaries (see Hoaglin, Mosteller and Tukey [104]): the mean appears as a circle, the median as a horizontal line separating the upper (third) and lower (first) quartile and the outliers are at the ends of the vertical dotted lines. Additionally, whenever a variant of the trade-off-based policy appears to differ only slightly in terms of performance with respect to a competitor, a Mann-Whitney \( U \) test ([164], also known as Wilcoxon rank-sum test) is performed to establish the significance of the performance difference, which is shown in separate tables. Following the conclusions of Sect. 2.17, the robustness metric is assumed as the aggregation of the positive and negative partial deviations of the makespans, treated and normalized in the same manner as the mean in Eq. 3.12 (see also Sect. 2.17). For the sample-based robustness metrics the following equations are assumed to hold:

\[
\sigma^2_{+}[C_{\text{max}}(\pi)] = \frac{1}{N} \sum_{i=0}^{N} [C_{\text{max}}(i, \pi) - \mu_n(\pi)]^2 1_{\{C_{\text{max}}(i, \pi) - \mu_n(\pi) > 0\}} \quad (3.13)
\]
Table 3.1: Outline of the experiments for the $P/Q|v_i, c_{ij}, \sim stoch, prec|E[C_{max}]$ problem, continued. [L] represents a level or level-like task attribute (see Sect. 2.16.1): B for a b-level, T for a t-level, D for dominant sequence. [NORM] represents a normalization (speed scaling) strategy: AVG for average speed (as in HEFT/SHEFT), MAX for maximum speed, NONORM for unnormalized (nominal) speed. [INS] is an insertion strategy: OPT for optimistic, BF for best-fit, empty otherwise (default, first idle time slot with an acceptable trade-off).

$$\sigma^2[C_{max_n}](\pi) = \sigma^2_+ [C_{max_n}](\pi) + \sigma^2_- [C_{max_n}](\pi)$$  \hspace{1cm} (3.14)
Chapter 3. Offline Scheduling of DAG-reducible Stochastic Workflows

The measured performance is shown with respect to the following independent parameters (horizontal axis): the processor multiplier ($p_{mul}$), the computation-to-communication ratio ($ccr$) and the normalized standard deviation ($nsd$) of the stochastic variables describing the duration of the activities in the application.

Note that the term $nsd$ refers here always and only to the normalized standard deviation (or coefficient of variation), prescribed for the stochastic variables describing the duration of the jobs and communication delays in the workflow. Thus, the $nsd$ is one of the inputs and is shown on horizontal axes. It should not be confused with the standard deviations of the normalized makespans (referred to as $\sigma^2_{n\rightarrow n}[C_{max}](\pi)$), which are outputs and, at the same time, robustness metrics of the scheduling policies. They are shown on the vertical axes, and may assume any value in $\mathbb{R}_{+/-}$.

The scheduling policies examined in the first experiment differ in their level of greediness and the manner of handling available jobs of lower priority in the presence of jobs of higher priority, that cannot be scheduled yet, at a given step. Three LAZY and three greedy (GDY) policies are presented, differing in the adopted prioritizing heuristics: -B, -T, -D for b-level, t-level and dominant sequence, respectively (see Sect. 2.16.1). In addition, the idle time-charged (ITC*) policy, shown in Alg. 5 and discussed in Sect. 2.16, is included. The variant reported here employs the dominant sequence priority heuristic. Its parameter driving the cost of accumulated idle time is set to one. As in preliminary measurements the performance of ITC heavily depended on the prioritizing method and parametrization chosen, only the best-performing ITC policy is reported. All of the above policies are thoroughly discussed in Sect. 2.16. The sole exception is the policy proposed in this chapter: TOFF/$\epsilon$-M. Two versions are evaluated, both using dominant sequence priorities: one with a fixed trade-off threshold value of 0.01, indicated on the result graphs as TOFF-$E=0.01$, and one with a variable threshold of $0.1/p_{mul}$, indicated as TOFF* for short.

In the second set, different possible estimation methods for the stochastic variables describing the activities in the workflows are compared, in order to determine the most accurate estimates. The suffix MM indicates the marginal mean (the “plain” expected value) estimate, MSD (for mean plus standard deviation) is the estimate proposed in the SHEFT algorithm ([227]) and PM (for partial moments) is the offline partial-moment-based estimate proposed in Sect. 3.5.1.
3.6. Experimental Evaluation

In the third set, scheduling policies with different priority determination and machine assignment heuristics are examined. The performance of the proposed TOFF* policy is compared with that of the state-of-the-art algorithms for heterogeneous environments: HEFT [23], SHEFT [22] and DLS [22]. DLS employs the dynamic level attribute and may only be based on this attribute: see Sect. 2.16.1. For both HEFT and SHEFT, all of the well-established prioritizing methods (b- and t-level, dominant sequence) are plugged in and tested, which is indicated by the first suffix, as this provides additional insights into the performance evaluation of these algorithms. In terms of stochastic estimates, for HEFT, the mean-based ones are used (MM), for SHEFT, the algorithm’s estimates of choice (MSD) and, in DLS, both are tested, which is indicated by the second suffix.

The fourth experiment set investigates the effects of the choice of different speed normalization strategies on the performance of TOFF*, with the parametrizations outlined for set 1, when the scheduling platform contains non-homogeneous processors and communication channels. The AVG strategy assumes average activity times across the available processing units and communication channels, as in both HEFT and SHEFT. The MAX strategy is based on the worst-case assumption of maximum activity execution times. The last strategy, NONORM, employs nominal (intrinsic, unnormalized) activity times. The aim of this part of the experiment is to determine which speed normalization method, and in which parameter ranges, delivers the lowest expected makespan in a heterogeneous environment, when applied to the best-performing algorithm, TOFF*. In addition, the effects of alternative normalization methods are examined on the robustness of the solutions.

In the fifth and final set, the insertion mechanism is investigated in more detail. As it has been outlined, TOFF* performs an insertion of a task of lower priority into the first idle time slot between two tasks of higher priority in which insertion is both feasible and unlikely to delay excessively any of the tasks of higher priority. This is the default insertion approach indicated on the graphs as TOFF* with no suffix. This approach, however, raises the question whether particular alternative insertion strategies could be better, at least in some parameter ranges. The optimistic insertion strategy (indicated by the OPT suffix) assumes that a job can fit in an idle time slot if the length of the slot is not smaller than the speed-scaled minimum job execution time. The minimum is, in turn, derived from the known distribution bounds. The BF (best-
3.6.1 Discussion

In comparison with the reference greedy and lazy policies in the first experiment set, both trade-off-based algorithms perform better in all considered parameter ranges and are more robust in most of the range of the normalized standard deviation of the inputs ($nsd$). The fixed-threshold variant, TOFF-E=0.01, is slightly inferior to its dynamic-threshold counterpart, TOFF*, but the measured difference in performance is only significant in low computation-to-communication ranges ($ccr < 1$) and in the high $nsd$ range ($nsd > 1$). The best-performing scheduling policy in the whole set, TOFF*, is only slightly less robust than a single competitor: the lazy algorithm employing the dominant sequence prioritizing scheme (LAZY-D). This is also true exclusively in an “almost deterministic” setting, i.e., for normalized standard deviation values close to zero. At the same time, LAZY-D does not perform consistently well throughout the whole $nsd$ range, nor is it a good choice in general, even among lazy algorithms: for example, it is the second-worst algorithm in high processor multiplier ranges ($p_{mul} > 0.6$). Similarly, other reference policies which deliver reasonably good results for particular parameter value ranges, such as the GDY-D algorithm at high computation-to-communication ratios ($ccr > 1$), are not consistent and fall behind their greedy, lazy or trade-off based competitors in this set. It is interesting to note that the best-performing idle time-charged algorithm, ITC*, is in fact among the most robust algorithms throughout the examined $nsd$ range. However, this robustness mostly comes at the price of clearly sub-optimal performance, except in settings with low $ccr$ values ($ccr < 1$). Overall, it may be observed that a controlled greediness approach brings decisive advantages with respect to purely greedy or lazy policies, but it must be a precisely guided approach, as in the case of TOFF, not an indiscriminate one, as in ITC.
3.6. Experimental Evaluation

Figure 3.5: Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s critical path (CP), as a function: (a) of the processor multiplier $p_{mul}$, given $|P| = \lceil \nu p_{mul} \rceil$; (b) of the computation-to-communication ratio ($ccr$). Algorithms with variable greediness.
Chapter 3. Offline Scheduling of DAG-reducible Stochastic Workflows

Figure 3.6: (c) Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation \( nsd = \frac{\sigma}{\mu} \) of the activity durations; (d) standard deviations of the normalized response time, as a function of \( nsd \). Algorithms with variable greediness.
3.6. Experimental Evaluation

<table>
<thead>
<tr>
<th>pmul</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-value</td>
<td>0.0977</td>
<td>0.0279</td>
<td>0.0170</td>
<td>0.0540</td>
<td>0.0392</td>
<td>0.0665</td>
<td>0.6919</td>
<td>0.4779</td>
<td>0.0063</td>
<td>0.7869</td>
</tr>
<tr>
<td>significance</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ccr</th>
<th>0.1</th>
<th>0.2</th>
<th>0.5</th>
<th>1</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-value</td>
<td>&lt;10^{-5}</td>
<td>&lt;10^{-5}</td>
<td>0.7501</td>
<td>0.6150</td>
<td>0.0083</td>
<td></td>
</tr>
<tr>
<td>significance</td>
<td>***</td>
<td>***</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>nsd</th>
<th>0.01</th>
<th>0.25</th>
<th>0.5</th>
<th>0.75</th>
<th>1</th>
<th>1.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-value</td>
<td>0.058</td>
<td>0.18</td>
<td>0.0063</td>
<td>0.91</td>
<td>0.0081</td>
<td>0.0293</td>
</tr>
<tr>
<td>significance</td>
<td>**</td>
<td>**</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td>*</td>
</tr>
</tbody>
</table>

Table 3.2: Statistical significance of the performance difference between TOFF* and TOFF-E=0.01. Two-tailed Mann-Whitney U test: sample size $5 \cdot 10^4$; * = significance at $p < 0.05$, ** = significance at $p < 0.01$, *** = significance at $p < 0.001$.

The second result set illustrates the performance and robustness of the TOFF* policy in isolation when employing different stochastic estimators. The estimator of choice, introduced in this chapter and based on the notion of partial moments (TOFF*-PM), consistently results in the best performance, except for high computation-to-communication ratios ($ccr > 1$), where the difference is less pronounced. It also delivers the most robust results, with the exception of the lowest range of the normalized standard deviation of the inputs ($nsd < 0.5$). At the same time, it must be noted that the design of TOFF* makes it particularly sensitive to the accuracy of the employed stochastic estimation mechanism. This makes unsuitable less accurate approximation methods, such as marginal means, as in the case of TOFF*-MM, or sums of means and non-directional standard deviations (TOFF*-MSD), even if these methods work reasonably well for other algorithms (see next set). Overall, for TOFF*, the MSD estimator still does perform better than MM, but this superiority has a narrow margin and is mostly visible in the upper processor multiplier range ($pmul > 0.4$).

In the third experiment set, the performance of the trade-off-based approach proposed in this chapter is compared with state-of-the-art heuristic policies, employing different prioritizing, machine assignment and stochastic estimation methods. This includes the HEFT and SHEFT algorithms with their default priorities (b- and t-level) and stochastic estimators (MM for HEFT and MSD for SHEFT) and the DLS algorithm with the dynamic level and marginal mean (MM) estimator. In order to provide further insights with respect to the experiments conducted in previous works reported in this chapter, for both HEFT and SHEFT the dominant sequence prioritizing method, and for DLS the MSD estimator, have been tested as well.
Chapter 3. Offline Scheduling of DAG-reducible Stochastic Workflows

Figure 3.7: Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s critical path (CP), as a function: (a) of the processor multiplier \( p_{\text{mul}} \), given \( |P| = \lceil vp_{\text{mul}} \rceil \); (b) of the computation-to-communication ratio \( (ccr) \). The proposed trade-off-based algorithm with different stochastic estimators.

108
3.6. Experimental Evaluation

Figure 3.8: (c) Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $nsd = \frac{\sigma}{\mu}$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $nsd$. The proposed trade-off-based algorithm with different stochastic estimators.
Chapter 3. Offline Scheduling of DAG-reducible Stochastic Workflows

Figure 3.9: Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s critical path (CP), as a function: (a) of the processor multiplier \( p_{\text{mul}} \), given \(|P| = \lceil vp_{\text{mul}} \rceil\); (b) of the computation-to-communication ratio (\( \text{ccr} \)). Algorithms with different job prioritizing and machine assignment heuristics.
3.6. Experimental Evaluation

Figure 3.10: (c) Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $n_{sd} = \frac{\sigma}{\mu}$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $n_{sd}$. Algorithms with different job prioritizing and machine assignment heuristics.
Chapter 3. Offline Scheduling of DAG-reducible Stochastic Workflows

The TOFF* policy outperforms all competing algorithms not only when the latter employ their default prioritizing and stochastic estimation methods, but also when alternative priority assignment and estimation strategies are used. In the lower normalized standard deviation range \((nsd < 0.75)\), HEFT-T and SHEFT-T perform closely to TOFF*, but are also markedly less robust. The same is true of the combination of the DLS algorithm and the MSD estimator in the higher processor multiplier range \((p_{mul} > 0.5)\). However, none of these algorithms are consistent in their performance in general, when compared to TOFF*. DLS-MM is slightly more robust than its better-performing counterpart which uses the MSD estimator. In almost deterministic settings \((nsd < 0.25)\), TOFF* is slightly less robust than HEFT-B/D, SHEFT-B/D and DLS-MSD, but the difference is not significant.

In the same setting, as well as throughout the examined \(nsd\) range, TOFF* still performs significantly better in terms of the mean normalized makespan than these and other competing algorithms in this set. TOFF* also outperforms the competitors by a much larger margin in the lower \(ccr\) range (for \(ccr < 5\)) and in the lower \(p_{mul}\) range \((p_{mul} < 0.5)\), making it a good candidate policy for computation-intensive scheduling problems, especially with scarce processing resources.

The fourth set of experiments investigates in more detail the performance and robustness of TOFF*, when different processor and channel speed normalization strategies are applied to task prioritizing on a heterogeneous scheduling platform. The unnormalized strategy, referred to by the suffix NONORM, consists in computing job priorities solely based on the intrinsic execution times of the relevant activities, as determined by the stochastic estimator employed (PM for TOFF*). This is by far the best strategy in the reported results. The only other strategy that has a comparable performance is MAX, which assumes maximum estimated activity times on the platform, i.e., behaves as if the activities were always scheduled to the slowest processors and communication channels. TOFF*-MAX, however, has a performance equal to that of TOFF*-NONORM only for restricted values of the middle range of the processor multiplier \((p_{mul} \in \{0.6, 0.8\})\), in the high range of the computation-to-communication ratio \((ccr \geq 1)\) and in the low range of the normalized standard deviation of the inputs \((nsd \leq 0.5)\).
3.6. Experimental Evaluation

Figure 3.11: Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s critical path (CP), as a function: (a) of the processor multiplier $p_{mul}$, given $|P| = \lceil vp_{mul} \rceil$; (b) of the computation-to-communication ratio (ccr). The proposed trade-off-based algorithm with different processor and channel speed normalization strategies.
Chapter 3. Offline Scheduling of DAG-reducible Stochastic Workflows

Figure 3.12: (c) Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation \( \text{nsd} = \frac{\sigma}{\mu} \) of the activity durations; (d) standard deviations of the normalized response time, as a function of \( \text{nsd} \). The proposed trade-off-based algorithm with different processor and channel speed normalization strategies.
3.6. Experimental Evaluation

<table>
<thead>
<tr>
<th>( p_{mul} )</th>
<th>0.1</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )-value</td>
<td>0.0486</td>
<td>0.0028</td>
<td>0.0016</td>
<td>0.0985</td>
<td>0.0490</td>
<td>0.85</td>
<td>0.0472</td>
<td>0.0637</td>
<td>0.0303</td>
<td>0.0648</td>
</tr>
<tr>
<td>significance</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( cer )</th>
<th>0.1</th>
<th>0.2</th>
<th>1</th>
<th>5</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )-value</td>
<td>0.0017</td>
<td>0.0270</td>
<td>0.0033</td>
<td>0.0908</td>
<td>0.0023</td>
</tr>
<tr>
<td>significance</td>
<td>**</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( nsd )</th>
<th>0.01</th>
<th>0.25</th>
<th>0.5</th>
<th>0.75</th>
<th>1</th>
<th>1.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )-value</td>
<td>0.0002</td>
<td>0.0036</td>
<td>0.0030</td>
<td>0.0075</td>
<td>0.0044</td>
<td>0.0526</td>
</tr>
<tr>
<td>significance</td>
<td>***</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
</tbody>
</table>

Table 3.3: Statistical significance of the performance difference between TOFF*/NONORM and TOFF*/MAX. Two-tailed Mann-Whitney \( U \) test: sample size \( 5 \cdot 10^4 \); * = significance at \( p < 0.05 \), ** = significance at \( p < 0.01 \), *** = significance at \( p < 0.001 \).

As outlined in Sect. 3.4, the AVG strategy assumes average activity execution times on the given platform and is the default strategy in HEFT and SHEFT algorithms. This strategy is the worst in the reported experiment set, as it only achieves a performance comparable to that of NONORM for particular values of the processor multiplier (\( p_{mul} \in \{0.2, 0.4\} \)). Admittedly, TOFF*-AVG is the most robust policy in most of the \( nsd \) range (except for \( nsd = 0.75 \)). However, a significant gain in terms of robustness is achieved by adopting the AVG strategy only in an almost deterministic setting (\( nsd = 0.01 \)). At the same time, AVG is outperformed by both NONORM and MAX by the largest margin in that particular setting. Overall, the best performing TOFF*-NONORM policy is also not significantly less robust than any competitor except in the above mentioned case.

The fifth and final set examines strategies used for the insertion of jobs into the idle time slots created by previous scheduling decisions. The default insertion method consists in inserting a task in the first idle time slot, for which an insertion is feasible and the trade-off value associated with the insertion is acceptable. Alternatively, the job in question is scheduled with no insertion. For comparison purposes, the best-performing greedy and lazy algorithm from set 1 has been chosen for each parameter value separately (GDY* and LAZY*, respectively). The TOFF* policy which employs the default insertion method clearly outperforms not only GDY* and LAZY*, but also other versions of TOFF, which apply optimistic (OPT-suffix) and best-fit (BF) insertion. The OPT strategy inserts a job in the first idle time slot that may be large enough for it. This strategy is consistently the worst, indicating that the associated number
of cases in which priorities are violated is too large for a reasonably good scheduling performance.

Figure 3.13: Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s critical path (CP), as a function: (a) of the processor multiplier $p_{mul}$, given $|P| = \lceil vp_{mul} \rceil$; (b) of the computation-to-communication ratio ($ccr$). Algorithms with different insertion strategies.
3.6. Experimental Evaluation

Figure 3.14: (c) Mean and dispersion of the makespan for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $nsd = \frac{\sigma}{\mu}$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $nsd$. Algorithms with different insertion strategies.
In addition, unlike many algorithm variants examined in other experiment sets, OPT does not compensate for poor performance with greater robustness. The BF strategy, on the other hand, looks for an idle time slot whose estimated length is the closest to the estimated execution time of the job chosen for insertion. This idea has been applied quite successfully in bin-packing-inspired scheduling in deterministic settings. However, it does not seem to bring any advantage in stochastic DAG scheduling: TOFF*-BF is consistently the second-worst performer in the simulation. This is likely due to the fact that the search for the idle time slots satisfying the requirements of the BF strategy pushes the insertion too much forward, as the first “acceptable” slots are skipped sometimes. This results in deferring the scheduling of the job considered for insertion and, indirectly, in a performance degradation.

3.7 Summary

This chapter addressed the problem of statically scheduling an application, given in form of a stochastic directed acyclic graph of tasks, with known probability distributions of computation and communication times. The problem setting is well justified, where static schedules, known in advance, are required. This is the case in computing environments with early reservations or with no monitoring or no practicable mechanism to switch processors at runtime, in project management and business workflows with additional requirements such as task-specific configuration, and in dedicated hardware solutions with fixed data flows.

The problem specifically targeted is $P/Q|v_i, c_{ij} \sim \text{stoch}, prec|E[C_{\text{max}}]$. An efficient heuristic approach is proposed to solve the problem, outperforming competing algorithms in an extensive set of simulations. It consists of two main components.

The first is a novel, more accurate yet computationally efficient method of estimating the values of the stochastic variables that describe the durations of the activities in the workflow. Such estimates are used both in the priority list determination phase and during the scheduling process, to derive the most probable start and finish times of the jobs in the workflow and thus drive the scheduling policy. This estimation method is based on the notion of partial moments, previously never applied in the context of scheduling.
The second component is an original trade-off mechanism, that is applied in the machine assignment phase. It is employed in those scheduling steps in which there are jobs of lower priority which are likely to be ready for execution within idle time slots in the schedule created by previous decisions: Due to unsatisfied dependencies, some of the jobs of higher priority might have already been scheduled for a later execution. In this approach, the (potentially beneficial) greedy scheduling of lower priority jobs is simulated. Then, on the basis of the stochastic estimates mentioned above, it is possible to estimate the delay that could be caused (local loss) for the higher priority jobs and the time gained (local gain) by the lower priority jobs. The trade-off in question expresses the potential loss with respect to the gain, considering also the relative priorities of the jobs. The proposed policy uses a parameter-driven approach to determine the acceptable trade-off value, which conceptually corresponds to a controlled greediness strategy. The complexity of this heuristic is still polynomial in the size of the inputs.

The experimental evaluation was based on a procedure that generates a large number of application models, where each is a directed acyclic graph with a random structure. The procedure then assigns completely arbitrary continuous four-parameter Beta distributions to the activities in the application. The evaluation demonstrated that the policy put forward in this chapter outperforms all competing algorithms not only when the latter employ their default prioritizing and stochastic estimation methods, but also when alternative priority assignment and estimation strategies are used. The difference is more pronounced in computation-intensive scheduling problems, especially with scarce processing resources, and less pronounced in almost deterministic settings, i.e., when the uncertainty of the inputs is small. With the exception of such settings, the proposed policy is also markedly more robust than its competitors. The estimator based on the notion of partial moments results consistently in the best performance, except for high computation-to-communication ratios, where the difference is less significant.
Chapter 4

Online Scheduling of DAG-reducible Stochastic Workflows

This Chapter investigates the problem of scheduling multiple instances of a workflow. As in the previous chapter, the workflow schema (application model) is assumed to have a structure in form of an arbitrary directed acyclic graph (DAG), where the nodes represent atomic tasks and the edges represent communication (data transfer) between connected tasks, which is required if these tasks are assigned to different processors. Both the tasks and the communication activities are examined in the time domain, i.e., as processing time and transfer delay, respectively. All these activity times are assumed to be described by stochastic variables, with arbitrary, known probability distributions.

The workflow instances are submitted into the system by a stochastic arrival process, that gives rise to a queue. The arrival process is not required to follow any specific distribution.

The scheduling platforms examined are homogeneous and heterogeneous computing environments, with a bounded number of identical or uniform processors, respectively, fully connected by identical or uniform physical channels.

Unlike in the previous chapter, it is assumed here that real-time monitoring and dispatching of tasks is allowed, as there are no additional constraints which would prevent it. Thus, it is possible to use an online approach to both the determination of task priorities and to the machine assignment phase. The fact that priorities do not need to be fixed in an up-front manner makes it possible to update the job priority lists dynami-
cally, i.e., based on (and thus taking advantage of) information on the actual activity execution times, becoming available at run time.

Unlike in most “classical” scheduling approaches that minimize the schedule length (see also Chapter 3), here the additional aim is to minimize the total time a workflow instance spends in the system, as perceived by the end user, i.e., the total response time, which is the sum of the expectations on the schedule length (makespan) and in-queue waiting time. The linear structure of this composite objective is additionally exploited by heuristic methods built into the proposed scheduling policies. Secondly, unlike in many contemporary cloud-oriented schedulers, provider-oriented criteria such as resource utilization (maximization) are also not of direct interest.

The following contributions are put forward:

1. a general online scheduling policy for the problem of minimization of the expected response time, when a queue of workflow instances is scheduled onto either a homogeneous or uniformly heterogeneous platform, i.e., the $G/G/c : P/Q | v_i, c_{ij} \sim \text{stoch, prec} | E[W]$ problem (see 2.12); the policy is parametrized, so that different (online and offline) job priority determination strategies and estimates of the stochastic variables of interest can be plugged in,

2. a method of improving the accuracy of the estimates of stochastic variables, based on the employment of conditional partial moments of the probability distributions for the determination of the duration of currently running activities, which results in an improvement of the expected makespan sub-objective at the cost of an adjustable, linear increase in complexity,

3. an optional local look-ahead subroutine to minimize the waiting time sub-objective by allowing incoming (new) workflow instances to start before the termination of the instances being processed, when said termination is unlikely to be affected,

4. an evaluation of the solutions with respect to their robustness (see 2.17).

The performance of the proposed approach is tested against comparable state-of-the-art algorithms known from the subject literature at the time of this writing. Where applicable, some of these algorithms are adapted to work in a multi-instance setting. In addition, related makespan-centric algorithms are extended with mechanisms that can be adopted to minimize total response time.
Chapter 4. Online Scheduling of DAG-reducible Stochastic Workflows

4.1 Motivating Scenarios

Consider the scenario in Figure 4.1 that illustrates a typical three-layered web application and is similar to that of Figures 3.2 and 3.3 in Chapter 3. As before, the front-end web server is responsible for the application’s presentation and a back-end composed by an application server and data sources such as databases and web services. However, in this case all functional components do not require a pre-run configuration phase and are not subject to licensing constraints or restrictions in the access to proprietary hardware or software. Likewise, the communication channels need not be set up for specific data amounts or transfer paths. Thus there is no reason or requirement that any of them be statically deployed and run exclusively on their dedicated processors or communication channels.

Some applications traditionally enforce a logical separation of functionally distinct components and the atomic tasks that require the most processing power are usually deployed to the fastest machine. However, many practical applications nowadays actually take advantage of the fact that their components may be freely deployed to a number of physical or virtual machines. In the cloud environment, for example, one of the assumptions is that real-time provisioning to satisfy the application’s requirements is always possible, because of the scalability of resources [21]. Even outside of the cloud environment, many modern applications facing high demand or high availability requirements also employ real-time ad-hoc deployment. In this fashion, it is possible to implement designs such as high availability (HA, see Piedad and Hawkins [187]), which may imply an application-level failover with stand-by/hot-swap application components (see, e.g., Jayaswal [120]). The arguably most advanced and flexible technique that may be used to implement HA designs and simultaneously to guarantee faster response times and/or better resource utilization, is load balancing (LB). An excellent example is the Linux-HA project (see Robertson [199]), where both low and high-level HA and LB can be realized.

Most implementations of these concepts rely on a monitor (such as Heartbeat in Linux HA) and time-out mechanism to decide that a processor is either busy with a task or unavailable, rather than on estimates derived from the analysis of the application itself. Most commonly, they also rely on pseudo-schedulers such as round robin to decide where the job that must run next should be executed. Such an approach is, in fact, an
4.1. Motivating Scenarios

However, if the application’s structure (workflow schema) and the computation and data communication requirements of its atomic activities are known in advance and static, this knowledge, along with the knowledge of the scheduling platform, can be used in the online scheduling process in LB. This also motivates the proposal of an informed online scheduling policy as outlined in later sections of this chapter.

Let us also revisit the scenario from Figure 3.4 in Chapter 3. We may theorize that, in the case of our example visa application processing department, ideally, all of the employees have eventually passed the necessary training and obtained the clearance to perform any of the tasks associated with the visa application for any customer. The processing speed depends on the aptitude of the employee. Obviously, a particular client’s case (i.e., workflow instance) usually requires additional time for (some of) the case’s information to be transferred, if different employees are to work on related tasks within the same case. Notably, this fits exactly into the optional communication delay model used throughout this thesis.

In all of the above presented cases real-time monitoring and dispatching of tasks is possible, or already being applied in practice.

4.1.1 Motivation for Response Time Optimization

From the point of view of the customer (end user), one of the most important performance criteria is the application’s perceived response time. The end user has no direct interest in provider-side performance indicators such as economic profitability or resource utilization, as long they do not impact the client side efficiency. Moreover, the client is not concerned, and arguably should not even be made aware of, the specific implementation of the online non-clairvoyant model (see Sect. 2.13).
values of the processing and waiting times of the tasks that compose the application’s instance (266). The response time is also widely recognized as an important element of a service level agreement (SLA) in grid and cloud computing. Elnozahy et al. [69] propose, for power-aware cluster systems, an equivalent of the SLA that is primarily constrained by an acceptable response time. Verma, Ahuja and Neogi, for example, propose an algorithm called pMapper (power-aware mapper), aimed at minimizing the power consumption in cloud provisioning scenarios, subject to a fixed performance requirement. The performance index employed is defined, for each task, as the ratio of the response time and the task’s deadline [255].

4.2 Problem Formulation

As in all problems of the class \(|\text{prec}|\text{C}_{\text{max}}\), let \(G = (J, E)\), with \(J\) the set of nodes and \(E\) the set of edges connecting them, be a single directed acyclic graph (DAG) representing an application model, or workflow schema. The nodes represent atomic jobs and the edges represent precedence constraints between them. Each job has an associated intrinsic execution time and each edge between two tasks has an associated communication delay. We follow the classical assumption that a delay is optional, i.e., caused solely by an actual transfer of data between two separate processing units. This happens only when two task nodes connected by an edge are scheduled to two different processors.

In addition, let all the intrinsic execution times \(w(j_i)\) and communication times \(w(e_{ij})\) be described by random variables \(w(\cdot)\) (see Sect. 2.2.1), with known arbitrary probability distributions, continuous or discrete, whose first and second partial moments and partial conditional moments are defined and finite (see 3.5.1, 4.4.1).

Let there be a stochastic process describing the arrival of new workflow instances into the system: is not required to follow any specific distribution. Optionally, for some specific parametrizations of the proposed algorithms, it is assumed that the average arrival rate \(\lambda\) is known.

The scheduling platform consists of \(Q\) with \(|Q| < \infty\) parallel processors and \(Q \times Q\) communication channels connecting them. Both the case of identical and uniform processors and channels, i.e., both a homogeneous and a uniformly heterogeneous
environment are examined (see Sect. 2.2.2). In the latter, the effective activity runtimes are given by the product of the intrinsic activity times and the speeds of the processor or communication channel an activity is mapped (scheduled) to.

The atomic tasks of the instances in the system are allocated to processors and executed without preemption: a job cannot be interrupted and later resumed. A processor must be idle at the moment of the allocation: there are no per-processor (local) job queues. The communication channels are assumed to be able to broadcast (possibly, at different speeds) the output data of a job from the processor on which the job has been completed to all other processors, where these data are assumed to be cached at no further cost and without any volume or time limit.

The $G/G/c : Q|v_i, c_{ij} \sim \text{stoch. prec} | E[W]$ problem can be formulated as follows: minimize the expected total response time $E[W]$, over an infinite horizon (number of workflow instances), when a queue of instances is scheduled to the given platform under the above outlined assumptions. The total response time for a single instance is given by $W = C_{\text{max}} + W_q$, where $C_{\text{max}}$ is the makespan of that instance and $W_q$ is its in-queue waiting time. Notably, the heuristic policies proposed exploit the linear aggregation of the two objective functions of which the objective $W$, or $E[W]$, consists (see Sect. 2.8).

4.3 Related Work

A convincing study by Lawrence and Sewell [146] presents extensive experimentation to show that relatively simple sensible online scheduling policies yield better results in terms of the makespan objective (and many others) than the most elaborate static algorithm working with uncertain information. This important result is true for both independent jobs and task graphs with dependencies. As the authors argued, the solutions derived by offline policies quickly deteriorate as uncertainty grows, i.e., when the inputs on which the decisions were based originally (the activity execution times) are subject to perturbation. It does not matter how extensive the information extracted from the known probability distributions is, or how sophisticated the algorithm that handles this information might be. As pointed out by the authors, an optimal solution technique for a simplification or abstraction of a scheduling problem cannot be trivially
applied to the original, more complex problem. This statement is applicable to many other scheduling and, more broadly, optimization approaches that employ reduction. Essentially, a solution to a stochastic scheduling problem obtained by reducing it to a deterministic one is of limited value as a solution to the original problem.

Research in the area of stochastic scheduling has followed this thought pattern. It has yielded results in the form of optimal or near-optimal online policies for a set of interesting objective functions, formulated as expectations on random variables such as makespan (Coffman et al. [56]) or weighted completion time of a set of jobs (Rothkopf [201]). It has often been argued by prominent authors in this field (e.g., in the works of Pinedo [188] and of Möhring, Schulz and Uetz [173] and Skutella and Uetz [221]) that in stochastic settings, the problem naturally reduces to finding policies for online scheduling decisions, and should not be treated as a combinatorial optimization problem to be solved in an offline fashion.

Naturally, the application of online policies must be feasible and permitted by the problem setting. As outlined in Sect. 4.2 we assume that an online approach is feasible here, therefore we follow the reactive scheduling paradigm (see Sect. 2.13).

There are, however, certain generic methods and specific heuristic techniques that are valid and useful in stochastic settings and applied to online policies, even though they originate from offline (predictive) approaches to deterministic scheduling problems. This includes the list scheduling approach in general (see Sect. 2.16), the earliest finish time machine assignment heuristic and the level- and critical path-based prioritizing heuristics. The latter (in particular b- ant t-levels [83], dynamic level [220], dominant sequence [141]) have been discussed in detail in Sect. 2.16.1.

The policies mentioned so far both in the context of (online) stochastic scheduling and offline scheduling of directed acyclic task graphs have been designed to schedule a single workflow instance, i.e., a single precedence-constrained set of jobs. However in the problem at hand the processing resources form a continuously operated system that must be able to process a periodic inflow (a stochastic arrival process) of such workflow instances. Multiple instances with different arrival times and activity execution times co-exist in a queuing system. From this perspective, other approaches that tackle problems sharing this aspect with the outlined problem are surveyed in what follows.

Iverson and Özgüner [116] proposed an algorithm for the minimization of the aver-
4.3. Related Work

Average schedule length in a heterogeneous environment that processes DAG-type workflows arriving according to a Poisson process, i.e., the $M/G/c : Q|\text{prec}, e_{ij} |E[C_{\text{max}}]$ problem. However, in their case the workflow instances (DAGs) are assumed to be deterministic and identical. In their solution, there is no centralized scheduler; instead, each workflow instance is associated with a local scheduling entity that performs task assignment using a heuristic that combines the low-complexity offline heterogeneous Dynamic Level Scheduling heuristic [220] with a queuing policy. The latter uses network and machine load estimates to decide in which order to place the tasks to be scheduled in each of the local processor queues. Bender and Rabin [27] provided an online scheduler for multiple “competing” task graphs, under several structural restrictions, and optimized the utilization of the processor pool. A distinctive feature of their approach is the assumption of fixed estimates for the processing requirements of the tasks and of randomly varying processor speeds. In addition, their scheduler takes advantage of preemption.

Zhao and Sakellariou [275] also investigated the problem of competition between multiple directed acyclic task graphs (DAGs). They focused on the issue of fairness, which is defined in reference to the so-called “slowdown” of a task graph, when it must compete with other graphs for the same resources in the processing environment. The DAG’s slowdown is the ratio of its two makespans: one computed when the DAG can use all the processing resources in the platform and a second one, when the interference of other DAGs is considered. A solution is fair, if for all the task graphs such slowdowns are (almost) equal. The DAGs in this approach are assumed to be scheduled simultaneously in a batch: there is no explicit arrival process. Bittencourt and Madeira [32] also targeted fairness among simultaneously scheduled workflows, understood as an acceptable level of the average makespan across the workflow instance set. Similarly, N’takpé and Suter [181] investigated “un-fairness” and average makespans in a set of specific grid environments, when scheduling sets of deterministic DAG-shaped workflows simultaneously, without an explicit arrival process.

Slightly closer to the problem presented in this chapter is the work of Chen and Maheswaran [49], in which the performance of a range of heuristics was investigated, under a stochastic arrival process of jobs modelled as DAGs, which did not, however, include stochastic activities. The (grid) scheduling platform was a restricted set of actual, specific hardware configurations. The proposed heuristics were tested on this
platform in terms of percentage of (composite) jobs meeting their deadlines and the generated network load. Gallet et al. [79] defined the throughput of a heterogeneous scheduling platform as the average number of workflow instances processed per time unit in steady state of the system: this corresponds to the service rate in queuing theory terms. To maximize the throughput, they proposed a static deployment algorithm that provides precomputed task allocation schemes which are fixed for all workflow instances and based on deterministic task processing and communication times.

Stavrinides and Karatza suggested to optimize the job guarantee ratio (the number of jobs completed within the respective deadlines to the total number of job arrivals) [223], under the assumption of an arrival process of DAG-reducible workflow instances. The heuristic proposed was based on the idea of exploiting idle time slots left by the predictive scheduling of a (number of) workflow instance(s) for the assignment of tasks belonging to successive workflow instances. In a follow-up work, Stavrinides and Karatza [224] optimized the overall system performance, given the achieved precision of computation results, to the total number of job arrivals. In this case, a range of bin-packing methods are combined with the imprecise computation technique outlined by Lin, Natarajan and Liu [156]. The idea is to exploit roughly (although not probabilistically) defined idle time slots in a schedule. Here, its use is based on the premise of being able to compute a predictive schedule, i.e., a complete offline schedule, based on deterministic activity times within the task graph, successively adjusted through the use of imprecise computation techniques.

Zhu et al. [277] proposed an algorithm for multiple workflows (DAGs) in an optical grid, based on the idea of combining all the currently schedulable DAGs into a set of tasks and of a discipline governing the choice of the tasks, termed Serve-On-Time (SOT). SOT, as opposed to FCFS, acts on the priorities of the tasks regardless of their workflow instance of origin. Hirales-Carbajal et al. [103] examined multiple standard benchmark workflows and twenty-five scheduling heuristics in a dedicated grid environment. The problem was defined as one of scheduling multiple workflow instances in a batch. They performance metrics employed included: the makespan approximation factor, the mean critical path waiting time (mean difference between the instance’s completion time and the length of its critical path) and the critical path slowdown (the quotient of the instance’s completion time and the length of its critical path). They put forward algorithms that outperformed the reference ones in terms of these metrics.
4.4. Solution Approach

However, the workflows considered in this approach are composed of deterministic activities and no arrival process of workflow instances is considered.

Many other approaches have been proposed in the past few years, in the grid and, most recently and numerously, cloud environments, targeting typically grid (or cloud) system optima (or system-centric criteria [260]), measured by resource utilization or fairness ([275],[261]), instead of a workflow-centric criteria, like makespan or throughput, as in the case of the scheduling policies investigated in this chapter. Many feature an explicit stochastic arrival process of “workflow” instances but actually trivialize the related scheduling problem by abusing the concept of workflow to reduce it to sets of unstructured, atomic jobs, so-called bag of tasks (BoT): e.g., Li et al. [154].

4.4 Solution Approach

In Chapter 3 we proposed, for the job prioritization sub-problem in the list scheduling approach, to substitute complete moments of the probability distributions of interest by an appropriate choice of their partial moments.

In this chapter, the method is further developed. Namely, the idea is to improve the accuracy of estimating path lengths during the scheduling process performed by an online algorithm, i.e., while the workflow instances of interest are being executed. The online setting makes it possible to take advantage of the information becoming available gradually, in particular to exploit the fact that the scheduler knows which activities are still running and how long they have run so far. This, in turn, allows for the computation of conditional moments for the purpose of estimating the durations of the activities that are still running. The specific novelty of the presented approach is the employment of conditional partial moments: conditional expectations and positive semi-deviations, in our case.

The improved estimates are applied both in the job priority list determination method and in the assignment phase of the scheduling policy. For this component of the scheduling policy, a novel trade-off mechanism is also proposed, which is aware of the stochastic nature of the investigated setting and designed for online scheduling (Sect. 4.4.2). The online multi-instance setting makes the mechanism fundamentally different from the one tailored to the offline single-workflow approach (see Sect. 3.5.2).
Additionally, in the multi-instance problem with the response time objective, it is also reasonable to exploit heuristics, which optimize the in-queue waiting time of an instance: This is implemented through a guided priority manipulation mechanism (see Sect. 4.4.2).

### 4.4.1 Conditional Partial Moments in Path Length Estimation

The general relationships governing conditional probabilities and existing between conditional moments and partial moments (see §3.5.1) are summarized by the following theorem and the formulas derived from it.

**Theorem 3.** Let \( g(X) \) be a function of a random variable \( X \), with a cumulative density function \( F(x) \). The expectation of \( g(X) \) over a set \( A \) is given by:

\[
E[g(X)|X \in A] = \frac{E_A[g(X)]}{Pr[X \in A]} = \frac{\int_A g(x)dF(x)}{\int_A dF(x)} \tag{4.1}
\]

**Proof.** Given two events \( K = \{X \in A\} \), \( H = \{g(X) = x\} \), the proof follows from Bayes’ theorem:

\[
Pr[H|K] = \frac{Pr[H \cap K]}{Pr[K]} \tag{4.2}
\]

where \( Pr[H|K] \) is the conditional probability of \( H \), where the condition is given by the event \( K \), having probability \( Pr[K] \), and \( Pr[H \cap K] \) the probability that both events, \( H \) and \( K \), occur simultaneously.

The following result is a theorem due to Winkler et al. [263].

The \( n \)-th conditional moment of \( X \), where the condition is given by the event \( X \in A \), is given by:

\[
E_{-\infty}^{z}[X^n|X \in A] = E[X^n] - E_{-\infty}^{z}[X^n] + z^n Pr[X \in A] \tag{4.3}
\]

In particular, the conditional expectation of \( X \) over the interval \([z, -\infty)\), given \( A = \{X \geq z\} \), is given by:

\[
E_{z}^{+\infty}[X|X \geq z] = \mu_{z} = \frac{\int_{z}^{+\infty} xdF(x)}{\int_{z}^{+\infty} dF(x)} \tag{4.4}
\]
4.4. Solution Approach

Thus the conditional expectation of $X$, when it is known that $X \geq z$, is its partial expectation over the interval $[z, +\infty)$, normalized by the unconditional probability that $X \geq z$. The last term, and in general any term containing the integral $\int_{z}^{+\infty} dF(x)$, is simply the CDF of $X$ for values greater or equal to a value $z$, and therefore easy to compute, assuming the CDF is given. It can also be computed alternatively using the following identity.

$$\int_{z}^{+\infty} dF(x) = 1 - \int_{-\infty}^{z} dF(x) \quad (4.5)$$

For the purpose of finding accurate estimates of stochastic path lengths to be used in a dynamic critical path (or level-based) list scheduling approach, in the offline scheduler (Chapter 3) the use of positive semi-deviations was proposed. In the online scheduling approach, for activities being processed at the time, analogously to the conditional expectations, the conditional semi-deviation can be derived. We are still interested in positive semi-deviations, as in the offline algorithm. Both the conditional expectation and the conditional semi-deviation are derived for the condition that the activity run time is no less than a certain, known value. Clearly, this value is the actual runtime normalized by the speed of the processor or communication channel to which the activity was allocated. The following formula can then be applied.

**Definition 4.1.** The conditional positive semi-variance of $X$, given $A = \{X \geq z\}$, is:

$$\text{Var}_+[X|X \geq z] = \sigma_{+z}^2 = \frac{\int_{z}^{+\infty} (x - \mu_z)^2 dF(x)}{\int_{z}^{+\infty} dF(x)} \quad (4.6)$$

where $\mu_z$ is the conditional expectation of $X$, given $A$. The positive semi-deviation $\sigma_{+z}$ is the square root of the conditional positive semi-variance of $X$.

4.4.2 The Stochastic Online Trade-off Mechanism

The general outline of the stochastic online trade-off-based approach is shown in Alg. [10]. As in the case of the offline trade-off-based scheduler (see Chapter [3]), it is assumed that the job prioritizing heuristic employed by the algorithm is likely to be accurate. In most “classical” list scheduling algorithms, such a list is followed strictly when the choice is made of the job to be scheduled next is made. The global makespan
minimization problem, which is intractable in general (Sect. 2.14), is transformed into the locally defined, polynomial problem of minimizing the earliest, or expected earliest finish time (EFT) of the job of highest priority. Like the priority determination, the EFT is also only a heuristic approach in the generalized DAG scheduling problem: in particular, it does not provide performance guarantees.

Algorithm 10: A generic formulation of TOFF/$\epsilon$-M: the proposed trade-off-based policy, driven by a threshold parameter $\epsilon$, for the multi-instance setting.

1 input: an instance of $G/G/c: Q|\text{prec}, v_i, e_{ij} \sim \text{stoch}|E|W|$; a threshold value $\epsilon$; optionally a maximum queue length $N$

2 output: a feasible schedule

3 while maximum queue length not exceeded do

4 wait until a job becomes available

5 compute the job priority list $L$; the priority list is dynamic

6 while $L$ has unexamined jobs do

7 consider the next highest priority job $j$ in $L$

8 for each processor $p$ in the pool, compute the estimated earliest finish time mapping $m(j, p)$ and cache it in a table of potential mappings $M$

9 for all jobs $j'$ with a higher priority than $j$ and with a mapping $m^*(j', p')$ in $M$, estimate the delay caused to the finish time of $j'$ by scheduling $j$ according to $m(j, p)$ and vice-versa

10 compute a trade-off between the delays

11 if the trade-off is above the threshold, remove $m^*(j, p)$ from $M$ and consider another processor until the processor pool is exhausted

12 otherwise, find the best mapping $m^*(j, p)$ for $j$ in $M$ and remove the remaining ones

13 if $j$ can already be executed according to $m^*(j, p)$, schedule it so, remove $m^*(j, p)$ from $M$ and $j$ from $J$

At a given step, a job of higher priority with respect to the list, i.e., from the point of view of the global objective, may not be available for scheduling according to the EFT requirement, i.e., on the processor that minimizes its expected EFT. A lazy scheduling policy (see Alg. 2) would not schedule any job in this case, whereas a greedy one (see Alg. 1) would schedule the next highest priority job available according to its EFT. In the first case, deliberate idle time is introduced into the schedule, which is lost, from the local point of view of lower priority jobs, which could be executed immediately (greedily). In the second case, a lower priority job may later interfere with what would be the best partial schedule for the higher priority job. In the deterministic scheduling problem, following the order imposed by the priority list is more likely to result in a
4.4. Solution Approach

schedule closer to the optimum. In the stochastic setting, however, it is possible that
greedily scheduled lower priority jobs will not interfere with higher priority ones.

In general, the length of the delay potentially caused to the finish time of a higher pri-

ority job by greedily scheduling a lower priority one follows a probability distribution,
as does the gain in the finish time of the job with lower priority. These distributions are
much easier to compute exactly in most cases than in the offline scheduling approach,
but estimates are still required to avoid the \#P-complete problem of computing such
distributions in the worst case. Because of the obvious differences between the offline
and online scheduling approaches, the stochastic loss and gain is not due to insertion,
simulated or effective, but to a repeated look-ahead mechanism. Thus there is a funda-
mental difference in the manner the necessary estimates are derived. Also, the trade-off
mechanism, that balances the loss and gain and takes into account the relative priorities
of the jobs under exam, works in a different manner with respect to the one proposed
for the offline scheduling policy.

In more detail, the proposed online scheduling approach works as follows.

Whenever a new job instance arrives or the monitor reports the completion of a task at
an instant in time \( \tau \), for all processors that are idle, a decision is made which task to
assign to that processor (as in all online elementary policies, see Def. 2.19). Let \( S \) be
the state of the system at \( \tau \). Algorithm provides this generic (common) framework.
It is designed to be called in a loop by an execution monitor. The variable \( \pi \) stands for
any specific policy that determines the task priority list and machine assignment and
thus shapes the behavior of the algorithm as a whole.

The priority list is generated as follows. First, the workflow instances currently present
in the system are prioritized according to their arrival time (to the FIFO discipline).
Then, within each instance, any standard prioritizing strategy may be used, be it level-
or critical-path-based, static or dynamic, e.g., the dominant sequence method. Each of
them may use different estimates of the stochastic variables of interest, either trivial
(e.g., mean-based) or context-adjusted (standard deviation- or partial moment-based).

Lines 4-5 of Algorithm implement a short-circuit strategy to immediately schedule
the highest priority task \( j_{\text{max}} \), if available (ready for execution), on the fastest processor
\( p_{\text{max}} \), if it is idle at the moment of the call. Starting with \( j_{\text{max}} \) if it could not be
scheduled, otherwise with the next task in the list, in decreasing order of priorities,
for all the processors, estimated start and finish times of the currently considered task on the given processor are cached (Lines 9-11). These estimates are employed by Algorithm 13 to determine the best possible mapping \( m_{\text{min}} \) for the current job \( j \), taking into consideration other potentially schedulable jobs (see below): a null mapping is returned if no schedule for \( j \) exists yet. Finally, the current task is scheduled, if it is available at the moment of the call (Alg. 11, Line 13). It is also easy to prove that every job will be scheduled eventually: if it is not scheduled earlier (greedily), it will be, after the removal of other jobs causes it to become the job of highest priority.

**Algorithm 11:** The online trade-off-based policy TOFFc-M(\( \pi \), \( S \)) in detail.

```plaintext
while \( U \leftarrow \text{UnscheduledJobs}(S) \) not empty do
  \( j_{\text{max}} \leftarrow \text{MaxPriorityJob}(\pi, U) \)
  \( p_{\text{max}} \leftarrow \text{MaxSpeed}(Q) \)
  if \( j_{\text{max}} \) is available and \( p_{\text{max}} \) is idle then
    schedule \( j_{\text{max}} \) on \( p_{\text{max}} \) and remove it from \( U \) and \( S \)
  while \( U \) not empty do
    \( j \leftarrow \text{MaxPriorityJob}(\pi, U) \) \( \triangleright \) consider the next highest priority job
    \( M \leftarrow \{ \} \triangleright \) potential mappings for all jobs
    foreach processor \( p \in Q \) in non-decreasing order of speed do
      \( (ST, FT) \leftarrow \text{EstimateTime}(\pi, j, p, S, M) \)
      \( M \leftarrow M \cup \{ (j, p, ST, FT) \} \triangleright \) add a new potential mapping for \( j \) to \( M \)
    \( m_{\text{min}} \leftarrow \text{BestTOFFMapping}(\pi, j, S, M) \triangleright m_{\text{min}} = (j, p_{\text{min}}, ST_{\text{min}}, FT_{\text{min}}) \) is the best potential mapping, with known start/finish times and processor assignment
    if \( m_{\text{min}} \) exists and \( ST_{\text{min}} = \text{CurrentTime}(S) \) then
      schedule \( j \) according to \( m_{\text{min}} \)
      remove \( j \) from \( U \) and \( S \) and all mappings for \( j \) from \( M \)
    else
      remove \( j \) from \( U \) \( \triangleright \) consider another task
```

Unlike all offline scheduling algorithms, that are unable to estimate distributed start and finish times and are not designed to take advantage of information available at runtime, the proposed scheduler does not follow a strict priority list and is able to exploit runtime information, with different accuracy levels. To this end, a mechanism is provided to compute such an estimate that incorporates the actual task execution and inter-task communication times when available. In each subsequent call (Algorithm 11, Line 11; Algorithm 13, Lines 3, 17, 20) to the corresponding function EstimateTime typically more actual values become available.
The routine \textit{EstimateTime} is a straightforward implementation of Eq. 2.7 in Sect. 2.3.2. The values substituted into the equation are derived through the policy-dependent estimate selector \( \text{Weight}(\pi, S, w(\cdot)) \), where \( \pi \) is the policy driving the selection, \( S \) is the system state, \( (\cdot) \in J \cup E \) is an activity (task or edge) and \( w(\cdot) \) is the corresponding stochastic variable.

Algorithm 12: The stochastic estimate selector \( \text{Weight}(\pi, S, w(\cdot)) \).

The most straightforward offline estimate for a stochastic variable is, naturally, its marginal mean. Another such estimate may be the sum of the mean and standard deviation, as postulated by Tang et al. in [227] or the sum of mean and positive semi-deviation, as proposed already in this thesis 3.5.1. In Sect. 4.4.1, on the other hand, the use of conditional partial moments is proposed: they are, naturally, online estimates, returned, when available, by the policy-driven selector (in the last code line). Naturally, offline estimates still must be used in the absence of information on actual execution times, that is otherwise available through the execution monitor (Lines 6-7). As we compare means of \textit{intrinsic} activity weights (processing/communication times) with actual running times on a set of processors with different speeds, a normalization step is necessary (Lines 5, 7, 9).

The trade-off-based dynamic priority assignment mechanism in the online variant of the algorithm works as follows. In Algorithm 11 Lines 10-12, the estimated start and finish times for the currently considered task were computed under the assumption of greediness, i.e., \textit{without regard for previous estimates for all the higher priority tasks} that are not ready for execution and thus could not be scheduled.
Let us now consider Algorithm 13.

**Algorithm 13:** BestTOFFMapping($\pi, j, S, M$)

```plaintext
1 if IsOBN($j, S$) then ▷ if $j$ is an out-branch node
2 $CP \leftarrow$ CriticalPath($Instance(j), S$)
3 $(ST^*, FT^*) \leftarrow \max_{jCP \in CP} \text{EstimateTime}(\pi, jCP, p, S, M)$ ▷ determine the finish time of the CP
4 $m_{min} \leftarrow (j, p(m_{min}), ST^*, FT^*)$ ▷ return a mapping for $j$ that is synchronous with the finish time of the CP
5 else
6 $FT_{min} \leftarrow \infty$
7 $m_{min} \leftarrow \text{null}$ ▷ initialize the output
8 foreach mapping $m = (j, p, ST, FT) \in M$ do ▷ for each of the possible mappings for $j$
9     check $\leftarrow$ true
10     if $FT < FT_{min}$ then
11         foreach mapping $m' = (j', p, ST', FT') \in M \setminus \{m\}$ do ▷ for each previously cached mapping for another job $j'$
12             $l \leftarrow \text{Priority}(t); l' \leftarrow \text{Priority}(t')$
13             if $l \geq l'$ then ▷ if $j'$ is a job of lower priority than $j$
14                 $FT_{min} \leftarrow FT, m_{min} \leftarrow m$
15         else
16             schedule $j$ according to $m$ in a copy of the state $S^*$
17             $(ST^*, FT^*) \leftarrow \text{EstimateTime}(\pi, j, p, S^*, M)$ ▷ determine the expected start and finish time of $j$, if it is scheduled according to $m$
18             if $ST^* > \text{CurrentTime}(S)$ then ▷ if, according to $m$, $j$ cannot be started yet
19                 continue foreach loop
20             $(ST'^*, FT'^*) \leftarrow \text{EstimateTime}(\pi, j', p, S^*, M)$ ▷ determine the expected start and finish time of $j'$, if $j$ is scheduled according to $m$
21             $\Delta FT \leftarrow (FT^* - FT), \Delta FT' \leftarrow (FT'^* - FT')$
22             $\theta \leftarrow \frac{\Delta FT'}{\Delta FT}$ ▷ determine the expected finish time differences and derive the trade-off value
23             if $\theta > \epsilon(\pi)$ then ▷ if the trade-off value is too high
24                 check $\leftarrow$ false; break foreach loop ▷ test the next mapping for $j$
25         if check then ▷ the trade-off value for $m$ is acceptable
26             $FT_{min} \leftarrow FT, m_{min} \leftarrow m$
27 return $m_{min}$ ▷ the EFT mapping for $j$, for which the trade-off is acceptable
```

Scheduling $j$ does not have to cause a conflict with a future mapping for a task $j'$ of
higher priority (Lines 12-14) but it might do so (Lines 15-22). Only in the latter case, a
new estimate of the finish time of $j'$ is computed under the assumption that $j$ would be
scheduled now (look-ahead for $j'$: Line 16). $\Delta FT'$ is the estimated difference of finish
times for $j'$, between the case when $j$ is greedily scheduled now and the case when the
given processor is left idle waiting for $j'$ to be ready. The potential gain is expressed by
the improvement $\Delta FT$ in the finish time of $j$, should it be scheduled now. The original
priorities $l, l'$ act as discount factors to compute the priority-discounted potential gain
and loss risk, respectively, where the loss is a potential delay of the higher priority job
$j'$. The trade-off is expressed by $\theta$, which is compared to a threshold value $\epsilon$, which
is an input parameter of the policy. Clearly, higher values of $\epsilon$ make the behavior of
the parameter-driven policy greedier, causing it to coincide with that of the greedy
algorithm for a sufficiently large $\epsilon$, whereas for $\epsilon = 0$ the behavior is that of the lazy
algorithm. Sect. 4.5 provides results for different values of this parameter and points
out choices of threshold determination strategies that result to be effective for a broad
range of randomly generated workflows.

Additionally, and unlike in makespan-centric online and offline algorithms, the pro-
posed approach exploits the known decomposition of the expected response time ob-
jective into expected schedule length and waiting time. In a simplified formulation,
the tasks that do not impact the critical path of a workflow instance, and thus are less
relevant for the expected makespan of that instance, need not be scheduled to minimize
their earliest finish time. This is reflected in Lines 1-4 of Alg. 13. It is sufficient that
these tasks, typically the out-branch nodes (OBNs) of the graph, be scheduled to finish
at or slightly before (Alg. 13, Line 4) the estimated finish time of the latest critical-path
task (obtained in Algorithm 13, Line 3). Deferring these tasks is unlikely to harm the
makespan of the more privileged workflow instance (the instance that arrived into the
system earlier) but, by allowing to schedule the highest priority job(s) of another (less
privileged) instance sooner, reduces the waiting time of the less privileged instance.

4.4.3 Complexity Analysis

This section evaluates the time complexity of the improved estimation methods (Sect.
4.4.1) and of the online trade-off-based scheduling approach (Sect. 4.4.2). The exact
complexity ($\Theta$) is provided for the steps for which it is available, the worst-case ($\hat{O}$)
complexity for the remaining cases. As in Chapter 3, a modified notation is adopted as the standard big-O \( O \) notation is not suitable for multiple input variables (see Howell [109]). As shown in Sect. 3.5.3, the complexity of the computation of each partial moment of a single probability distribution is linear in the number \( s \) of the support points (see Sect. 2.9) in the discrete representation of its CDF. From this, the overall exact complexity of this step can be derived according to Eq. 3.11 (repeated below):

\[
\hat{\Theta} = 2sa = 2s(v + e)
\]  

(4.7)

For the purposes of the above equation, the total number of activities \( a = v + e \) does not change with respect to the offline scheduling context: \( v \) is still the number of jobs and \( e \) the number of the communication edges (precedence constraints) in the task graph of the application (workflow model) and is not related to the multiplicity of the instances. In other words, because the task graph and the associated probability distributions are static information, the complexity of the computation of partial moments is not influenced by the change of the algorithm’s setting to an online one, with a queue of (multiple) workflow instances.

The optional calculation of the conditional moments of activities that are already running, according to Eq. 4.4 and Eq. 4.6, requires, for each equation, two additional operations, associated with the computations of the integrals. The complexity of each of these operations is, however, only linear in the number \( s \). As in Sect. 3.5.3, \( s \) can be assumed as constant for all discrete CDF representations in the problem instance. However, the number of such additional operations is equal to the total number \( r \) of activities currently running in the system, which depends on the previous behavior of the scheduling algorithm in relation to all the input variables of a particular problem instance. The total complexity of the runtime information-driven priority list computation based on conditional partial moments is thus given by:

\[
\hat{\Theta}_{\text{cond}} = 2sa + 4sr = 2s(a + 2r)
\]  

(4.8)

The online list scheduling policy shown in Alg. 11 needs \( u \) steps to perform the mapping evaluation for the \( u \) jobs yet to be scheduled. Each such evaluation entails determining the best mapping for the job on each processor \( (pu \) operations), then computing
the trade-off values (in Alg. \[13\]) under consideration of the previously fixed mappings for every job of higher priority: there are at most \(u - 1\) such mappings. The worst-case complexity of the trade-off-based mapping evaluation driving the machine assignment phase is, therefore, given by:

\[
\hat{O} = pu(u - 1)
\] (4.9)

The total complexity of any list scheduling policy also depends on the complexity of the algorithm determining the priority list, which is external to the proposed estimation methods and machine assignment heuristic. Typical level-based prioritizing methods, such as b- and t-level, are worst-case quadratic, while the dominant sequence is worst-case cubic in the number of nodes in the task graph (see Sect. \[3.5.3\]). In an online multi-instance context, there may be many jobs of different instances to prioritize. However, given the assumption of a FIFO instance priority, the tasks belonging to different instances are naturally partitioned, with a number of partitions equal to the number \(n\) of the active instances in the system.

Thus the total worst-case complexity of the TOFF/\(\epsilon\)-M policy with conditional partial moment-based estimators and a dominant sequence priority list is given by:

\[
\hat{O} = nv^3 + 2s(a + 2r) + pu(u - 1)
\] (4.10)

The number \(r\) of the currently running activities is subject to the following conditions:

\[
r = r_v + r_e \leq na = n(v + e), \quad r_v \leq p, \quad r_e \leq p^2
\] (4.11)

The number \(u\) of the currently unscheduled jobs is given by:

\[
u \leq nv - r_v
\] (4.12)

The number \(n\) of active instances in the system depends on all of the following: the problem instance, including the realizations of all the arrival and activity execution times of all the workflow instances, the scheduling policy employed and the current time. Queuing theory provides some closed-form results for the number of instances
in the system at a particular moment (as a function of time), however, not for queues with a general departure process, such as the one under consideration. On the other hand, the long-term average number of instances \( E[n] \) is known and given by Little’s law (see Sect. 2.18):

\[
E[n] = \lambda E[W]
\]

where \( \lambda \) is the mean arrival rate and \( E[W] \) the expected response time. The term \( E[W] \), however, also depends on the problem instance, the realizations of all the arrival and activity execution times of all the workflow instances and the scheduling policy employed.

Finally, the total time complexity of TOFF/\( \epsilon \)-M can be approximated as:

\[
\hat{O}_{total} = nv^3 + 2s(a + 2r) + pu(u - 1) \leq nv^3 + 2s(v + e + 2p^2) + pn^2v^2
\]

Typically, online policies are more efficient than offline ones, given the shorter horizons and thus a smaller decision space they have to explore. In the proposed policy, in many cases the decision space is restricted by the observation that a locally optimal scheduling decision is immediately apparent (Alg. 11, Lines 5-6) or that computing a trade-off involving two estimate finish times both lying in the future is useless for the current scheduling decision (Alg. 13, Lines 18-19).

### 4.5 Experimental Evaluation

For evaluation purposes, the procedure outlined in Sect. 2.19 was employed. A set of 500 graphs was created and, for each single graph from the above set, 100 realizations, for a total of 50000 different workflow instances. Each of the 500 graphs has an arbitrary structure, randomly generated by a procedure adapted from the benchmark due to Kwok and Ahmad [142] and Topcuoglu, Hariri and Wu [233] and parametrized as follows:

- number of tasks \( v \in \mathbb{N} \), chosen randomly from the interval [10, 25],
4.5. Experimental Evaluation

- maximum alpha $\alpha_{\text{max}}$ of 0.1, 0.25, 1, 4, 10; the actual values of $\alpha$ uniformly distributed in the interval $(0, \alpha_{\text{max}}]$, where $\alpha \sqrt{v}$ is the mean graph height,
- maximum out-degree of a node of 10 and actual out-degree uniformly distributed in the interval $[1, \text{max}]$,
- maximum level jump (difference in levels in the graph between communicating tasks) of 5 or 10 and actual jump uniformly distributed in $[1, \text{max}]$,
- mean job execution times $w(\cdot)$ sampled uniformly randomly from $[1, 10]$,
- computation-to-communication ratio (CCR) of 0.1, 0.2, 1, 5, 10.

Note that only the mean job execution times $w(\cdot)$ are taken from a normal distribution with the given support. They are successively used to extrapolate completely arbitrary continuous four-parameter Beta distributions (see Appendix A) such that each distribution has a mean $\mu = w(\cdot)$ and is supported in $[\frac{\mu}{4}, 4\mu]$ and has a normalized standard deviation $\sigma_{\mu}$ of a value from the set $\{0.01, 0.25, 0.5, 0.75, 1, 1.25\}$. Analogously, the communication delays are assigned four-parameter Beta distributions extrapolated in a manner such that the requirement given by the CCR factor is satisfied.

Summarizing, each instance has the same structure and precedence constraints, but different activity execution times.

For each graph and number of processors, an estimate $\lambda_{\text{max}}$ of the critical arrival rate is computed in the pre-simulation phase. It is the maximum value of the arrival rate that does not result in an increasing queue of instances, when scheduling is performed with a lazy algorithm with a dominant sequence priority. The actual values of the arrival rate $\lambda$ are calculated as $\lambda = \lambda_{\text{mul}} \lambda_{\text{max}}$, where the factor $\lambda_{\text{mul}}$ is chosen from $\{0.25, 0.5, 0.75, 1, 1.25\}$.

Similarly, the number of processors $|Q|$ is defined as a set of multipliers $p_{\text{mul}} \in \{0.25, 0.5, 0.75, 1\}$ of the nodes $v$ of the task graphs as $|Q| = [p_{\text{mul}} v]$. The processor speeds are chosen as uniform random values in the interval $(0,1]$, as are the speeds of the processor-to-processor physical channels. For a particular processor, the speed of the channels connecting it to other processors is not correlated with its own speed.

To evaluate the effectiveness of the improved conditional partial moment-based estimation methods, of the trade-off-based policy design and of the active OBN delay
mechanism proposed in this chapter, the simulations outlined above are run against a number of well-established scheduling policies, grouped by relevant features. Table 4.1 provides the details.

The evaluation is also based on the methodology outlined in Sect. 2.19. In particular, the performance measure given in the equation below is chosen.

\[
E[W_n](\pi) = \frac{1}{N} \sum_{i=0}^{N} \frac{W(i, \pi)}{\sum_{j \in J_{CP}(i)} r(i, w(j))} \tag{4.15}
\]

For each scheduling policy \( \pi \), the values of the response time \( W(i, \pi) \) are determined, for \( N \) workflow instances. Each sampled response time, corresponding to an instance \( i \), is normalized to the overall time needed for the processing of the instance’s critical path (the sum \( \sum_{j \in J_{CP}(i)} r(i, w(j)) \) in Eq. 4.15). The performance metric for each policy is the mean of these normalized response times. Besides the mean \( E[W_n](\pi) \), shown on the graphs as a circle, for the normalized response time \( W_n(\pi) \), its statistical dispersion is also shown, given by five-number summaries ([104]). Additionally, whenever a variant of the trade-off-based policy appears to differ only slightly in terms of performance with respect to a competitor, a Mann-Whitney \( U \) test [164] is performed to establish the significance of the difference, which is shown in separate tables. The robustness metric is the aggregation of the positive and negative partial deviations of the response times, treated and normalized in the same manner.

The measured performance is shown with respect to the following independent parameters (horizontal axes): the processor multiplier (\( p_{mul} \)), the arrival rate multiplier (\( \lambda_{mul} \)) and the normalized standard deviation (\( nsd \)) of the stochastic variables describing the duration of the activities. Note that the term \( nsd \) refers here only to the normalized standard deviation (or coefficient of variation), prescribed for the stochastic variables describing the duration of the jobs and communication delays. The \( nsd \) is thus one of the inputs and is shown on horizontal axes. On the other hand, the standard deviations (positive and negative) of the normalized response times are outputs and, at the same time, robustness metrics of the scheduling policies. These are not normalized standard deviations of \( W \), but standard deviations of sampled values of \( W_i \ (i \in [1, N]) \), previously normalized each by the length of the \( i \)-th critical path. They are referred to by \( \sigma_{+/-}^2[W(\pi)] \), shown on the vertical axes, and may assume any value in \( \mathbb{R}_{+/-} \).
4.5. Experimental Evaluation

<table>
<thead>
<tr>
<th>Set</th>
<th>Feature(s)</th>
<th>Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Greediness</td>
<td><strong>LAZY</strong>: lazy algorithm (2), strict FIFO / dominant sequence</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>GDY-P</strong>: greedy algorithm (1), priority-driven: baseline FIFO and dominant sequence but no deliberate idle time</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>GDY-R</strong>: greedy algorithm (1), random LEPT/LVF job first</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>ITC-E=k</strong>: idle time-charged algorithm (5), cost-charging parameter set to k, baseline FIFO and dominant sequence</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>OBN</strong>: TOFF/ε-M with variable $\epsilon = 0.1/\lambda_{mul}$, baseline FIFO / dominant sequence, active OBN delay; average results for OBN with different estimators (see set 2)</td>
</tr>
<tr>
<td>2</td>
<td>Estimators</td>
<td><strong>OBN-MM</strong>: OBN (TOFF/ε-M, $\epsilon = 0.1\lambda_{mul}/p_{mul}$) with marginal mean estimation</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>OBN-MSD</strong>: OBN with SHEFT-like estimation (MSD: sum of marginal mean and standard deviation)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>OBN-PM</strong>: offline partial moments-based estimation</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>OBN-CM</strong>: online cond. partial moments-based estimation</td>
</tr>
<tr>
<td>3</td>
<td>Priority and machine assignment heuristics</td>
<td><strong>HEFT-[L]</strong>: baseline FIFO queue, job priority attribute [L] (see legend), marginal mean estimation</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>SHEFT-[L]</strong>: baseline FIFO, job priority attribute [L], MSD</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DLS-MM</strong>: baseline FIFO, marginal mean estimation</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>DLS-MSD</strong>: baseline FIFO, SHEFT-like estimation</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>OBN-CM</strong>: see set 2</td>
</tr>
<tr>
<td>4</td>
<td>Normalization strategy</td>
<td><strong>OBN-PM/[NORM]</strong>: OBN with offline partial moments-based estimation, normalization strategy [NORM] (see legend)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>OBN-CM/[NORM]</strong>: OBN with online conditional partial moments-based estimation, normalization strategy [NORM]</td>
</tr>
<tr>
<td>5</td>
<td>Cross-instance strategy</td>
<td><strong>LAZY</strong>: lazy algorithm (see set 1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>GDY</strong>: greedy algorithm, priority-driven only (see set 1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>OBN</strong>: see set 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>TOFF-NOOBN</strong>: “OBN” algorithm with the active OBN delay mechanism turned off</td>
</tr>
</tbody>
</table>

Table 4.1: Outline of the experiments for the $G/G/c : P/Q|v_i, c_{ij} \sim \text{stoch, prec}|E[W]$ problem. [L] represents a level-like task attribute: B for b-level, T for t-level, D for dominant sequence. [NORM] represents normalization (speed scaling): AVG for average activity time (as in HEFT/SHEFT), MAX for maximum activity time on the platform, NONORM for unnormalized (nominal) activity time.
In all the algorithms the natural FIFO instance ordering (queue discipline) in accordance with the arrival process is employed. It is always followed strictly by the LAZY algorithm (experiment sets 1 and 5). Given this assumption, different prioritizing heuristics may be employed within each instance, however only the best performing LAZY algorithm is shown: the dominant sequence-based one. All the other policies may choose to ignore the FIFO discipline in some of the scheduling steps, in the absence of available jobs belonging to the instance of highest priority (hence the specification “baseline FIFO priority” in Table 4.1). The sole exception is the GDY-R policy, which schedules, at any decision point, a virtually random job according to the LEPT/LVF rule, i.e., the job with the longest expected processing time (largest marginal mean), with tie-breaking based on the largest variance. GDY-P is a greedy policy that attempts to schedule jobs according to FIFO and dominant sequence (D) priorities, but it does not admit deliberate idle times: if the highest D-priority job is unavailable on any processor, the next available D-priority job is scheduled, or the highest D-priority job within the next instance. The ITC family of policies is based on the idle time-charged algorithm shown in Alg. 5, whereby the jobs are prioritized according to FIFO and dominant sequence and the parameter driving the cost of accumulated idle time is set as shown after the equality sign (see Table 4.1). All of the algorithms in the first experiment set have been thoroughly discussed in Sect. 2.16. The sole exception is a version of the policy proposed in this chapter: TOFF/ε-M, with a variable threshold of $0.1/\lambda_{mul}$, a baseline FIFO and dominant sequence priority and an active out-branch-node (OBN) delay mechanism, which is hence indicated on the result graphs as OBN for short.

The first set of experiments comprises scheduling policies with different greediness: the aim is to examine how the proposed OBN policy performs in comparison to greedy algorithms (“myopic”, as GDY-R, and “clever”, as GDY-P) on the one hand, and lazy algorithms on the other. OBN, briefly characterized, is an algorithm that exhibits a greedy or lazy behavior, based on the values of its input parameter $\epsilon$. Other approaches to parameter-driven, or controlled greediness, represented by the ITC family of policies are naturally included for comparison.

---

1Technically, GDY-R is implemented in a way that it still begins searching for schedulable jobs according to FIFO and dominant sequence priorities. However, given its design, it may deviate from the original priorities to an arbitrarily large extent.
4.5. Experimental Evaluation

In the second set, the different possible estimation methods for the stochastic variables describing the activities in the workflows are compared, in order to determine the most accurate estimate. In particular, the suffix MM stands for the most “obvious” estimate: the (marginal) mean, whereas MSD (for mean plus standard deviation) is the estimate proposed in the SHEFT algorithm ([227], see also [3.3]), the suffix PM (for partial moments) is the offline partial-moment-based estimate introduced in Sect. 3.5.1 and, finally, CM (for conditional partial moments) is the estimate proposed in Sect. [4.4.1].

The third set comprises scheduling policies with different priority determination and machine assignment heuristics. In this set, the performance of the proposed OBN policy is compared with that of the multi-instance versions of the state-of-the-art list algorithms for heterogeneous environments: HEFT [233], SHEFT [227] and DLS [220]. DLS employs the dynamic level attribute (and may only be based on this attribute: see Sect. 2.16.1), whereas, for both HEFT and SHEFT all of the well-established prioritizing methods (b- and t-level, dominant sequence) are plugged in and tested, which is indicated by the first suffix. In terms of stochastic estimates, for HEFT, the mean-based ones are used (MM), for SHEFT, the algorithm’s estimates of choice (MSD) and, in DLS, both are tested, which is indicated by the second suffix.

In the fourth set, different speed normalization strategies are examined that are applicable to the prioritizing and machine assignment phases in list scheduling, in the presence of non-homogeneous processors and communication channels in the platform. These strategies include: the average activity time-based one, as in both HEFT and SHEFT (AVG), one based on the maximum activity execution time (MAX) and one using the nominal (intrinsic) activity time (NONORM). The effects of the chosen strategy are tested by plugging it into the OBN scheduling policy, which is a version of TOFF/\(\epsilon\)-M with all the parametrizations and refinements outlined for set 1. The aim is to determine which speed normalization method delivers the best performance in a heterogeneous environment, given the response time objective and, in particular, whether the results obtained here differ from those relative to makespan minimization.

In the final set, the effectiveness of the active OBN delay mechanism proposed in this chapter is investigated separately. The OBN scheduler is compared with an identically parametrized version of TOFF/\(\epsilon\)-M, where the part of the algorithm responsible for the said mechanism is turned off (TOFF-NOOBN), as well as with the “plain” greedy and lazy policies (GDY, LAZY). The LAZY policy never schedules any job of an instance
of lower (FIFO) priority, as long as all of the jobs of an instance of higher priority have not been assigned. The GDY policy, on the contrary, in the absence of an available job of an instance of higher priority, always schedules the next highest priority job, belonging to any instance. TOFF-NOOBN also exhibits a greedy behavior, but only for scheduling decisions, for which the corresponding trade-off values are acceptable. Finally, the special provisions in OBN override the trade-off-based mechanism in TOFF such that any trade-off is accepted (the algorithm mimics GDY) when the finish time of the highest priority instance is unlikely to be delayed.

4.5.1 Discussion

With respect to the reference policies (“plain” greedy and lazy algorithms) in the first set, OBN generally both performs the best and is the most robust. The closest competitor is, quite surprisingly, the GDY-R policy, which, however, falls behind when there are relatively few processing units or when the nsd of the activity durations is low. For large relative sizes of the processor pool ($p_{mul}$) both greedy policies come close to the performance of OBN. At the same time, the performance of LAZY and “almost lazy” ITC algorithms (those with a high threshold value) worsens with larger $p_{mul}$, when compared to the others. In general, LAZY and ITC have performed the worst in this set, as their performance is also negatively impacted by growing instance arrival rates ($\lambda_{mul}$). Because $\lambda_{max}$ approximates the critical arrival rate for the LAZY policy, an arrival rate $\lambda \approx \lambda_{mul}\lambda_{max}$ and higher should cause an instability of the system under that policy. This is revealed by a large difference in the sample mean of the response time between a stable and an unstable system (here: $\lambda_{mul} \geq 1$). In an infinite queue, the true mean would be infinite in the latter case (see Sect. 2.18). For all values of $\lambda_{mul}$ OBN outperforms not only the lazy, but also the greedy competitors. However, the difference is much more pronounced for $\lambda_{mul} \in \{0.5, 0.75, 1.0\}$, while at both ends of the examined range it is smaller. OBN brings the most advantage, in terms of performance, in the low range of the values of the normalized standard deviation (nsd) of the activity times. However, it is comparatively robust even for much higher nsd values, up to and including nsd = 1. Beyond this value, on the other hand, none of the policies seem to be very robust. Again, the LAZY and ITC algorithms are markedly less robust in the high range of nsd than the rest.
4.5. Experimental Evaluation

![Graph](image)

Figure 4.2: Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s critical path (CP), as a function: (a) of the processor multiplier $p_{mul}$, given $|P| = \lceil vp_{mul} \rceil$; (b) of the multiplier $\lambda_{mul}$ of the critical arrival rate, given $\lambda = \lambda_{mul} \lambda_{max}$. Algorithms with variable greediness.
Figure 4.3: (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $nsd = \frac{\sigma}{\mu}$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $nsd$. Algorithms with variable greediness.
4.5. Experimental Evaluation

<table>
<thead>
<tr>
<th>( p_{mul} )</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )-value</td>
<td>0.0169</td>
<td>0.0169</td>
<td>0.0050</td>
<td>0.0654</td>
<td>0.0066</td>
<td>0.0050</td>
<td>0.0044</td>
<td>0.0074</td>
<td>0.0016</td>
</tr>
<tr>
<td>significance</td>
<td>*</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>( \lambda_{mul} )</th>
<th>0.25</th>
<th>0.5</th>
<th>0.75</th>
<th>1</th>
<th>1.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )-value</td>
<td>0.0052</td>
<td>0.0044</td>
<td>0.0064</td>
<td>0.0056</td>
<td>0.0072</td>
</tr>
<tr>
<td>significance</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>nsd</th>
<th>0.25</th>
<th>0.5</th>
<th>0.75</th>
<th>1</th>
<th>1.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>( p )-value</td>
<td>&lt;10^{-4}</td>
<td>0.0014</td>
<td>0.0072</td>
<td>0.078</td>
<td>0.092</td>
</tr>
<tr>
<td>significance</td>
<td>***</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
</tbody>
</table>

Table 4.2: Statistical significance of the performance difference between OBN and GDY-P. Two-tailed Mann-Whitney \( U \) test: sample size \( 5 \cdot 10^4 \); * = significance at \( p < 0.05 \), ** = significance at \( p < 0.01 \), *** = significance at \( p < 0.001 \).

The second result set shows that the partial- and conditional partial moment-based estimators are the best choice, especially for low relative sizes of the processor pool and higher relative arrival rates, i.e., in a situation, where the total number of jobs per time unit is large with respect to the total processing capacity of the system. For very large processor pools, the differences among all the estimators virtually disappear, as they do for low arrival rates. Wherever the performances differ more significantly, the closest competitor to the \( \text{CM} \) estimator is \( \text{PM} \). Notably, the \( \text{CM} \) estimator outperforms \( \text{PM} \) by a larger margin in settings with very small processor pool sizes and large normalized standard deviations of activity durations (\( \text{nsd} \)). \( \text{CM} \) is more robust than any competitor, with one case (for \( \text{nsd} = 0.25 \)), where \( \text{PM} \) is better in this respect. For very large \( \text{nsd} \) values, \( \text{CM} \) is the only estimator that delivers a comparatively robust outcome. The SHEFT-type \( \text{MSD} \) estimator is slightly better than the marginal mean in most cases, except for very large \( \text{nsd} \) values: in this case, \( \text{MSD} \) also delivers by far the least robust result.

When compared to the state-of-the-art HEFT- and DLS-based schedulers (third set), overall, the \( \text{OBN} \) policy with the best-performing conditional moment-based estimator (\( \text{OBN-CM} \)) delivers the best results in terms of both the objective function value and its statistical dispersion. For very small processor pool sizes and large activity standard deviations the online multi-instance variants of the DLS algorithm, regardless of the employed estimators, i.e., both \( \text{DLS-MM} \) and \( \text{DLS-MSD} \), come close to \( \text{OBN-CM} \). \( \text{DLS} \) is also slightly better in terms of robustness, but exclusively in the lowest part of the range of \( \text{nsd} \) values. The \( \text{HEFT} \) and \( \text{SHEFT} \) algorithms, in this setting, perform very similarly whenever the same priority list determination heuristic is employed: for
example, HEFT-B and SHEFT-B.

Figure 4.4: Sample mean and dispersion of the response time for an instance, normalized by the processing time of all the task realizations in the instance’s critical path, as a function: (a) of the processor multiplier $p_{\text{mul}}$, given $|P| = \lceil vp_{\text{mul}} \rceil$; (b) of the multiplier $\lambda_{\text{mul}}$ of the critical arrival rate, given $\lambda = \lambda_{\text{mul}} \lambda_{\text{max}}$. The proposed trade-off-driven algorithm with OBN delay, working with different stochastic estimators.
Figure 4.5: (c) Sample mean and dispersion of the response time for an instance, normalized by the processing times of the task realizations in the instance’s CP, as a function of the normalized standard deviation \( nsd = \frac{\sigma}{\mu} \) of the activity durations; (d) standard deviations of the normalized response time, as a function of \( nsd \). The proposed trade-off-driven algorithm with OBN delay, working with different stochastic estimators.
Chapter 4. Online Scheduling of DAG-reducible Stochastic Workflows

<table>
<thead>
<tr>
<th>$p_{mul}$</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1</th>
<th>1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$-value</td>
<td>0.0052</td>
<td>0.0181</td>
<td>0.00190</td>
<td>0.0079</td>
<td>0.4622</td>
<td>0.0903</td>
<td>0.0966</td>
<td>0.8595</td>
<td>0.0485</td>
<td>0.9244</td>
</tr>
<tr>
<td>significance</td>
<td>**</td>
<td>*</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>$\lambda_{mul}$</td>
<td>0.25</td>
<td>0.5</td>
<td>0.75</td>
<td>1</td>
<td>1.25</td>
<td>1</td>
<td>1.25</td>
<td>1</td>
<td>1.25</td>
<td>1</td>
</tr>
<tr>
<td>$p$-value</td>
<td>0.0288</td>
<td>0.0160</td>
<td>0.0876</td>
<td>0.0396</td>
<td>0.0166</td>
<td>0.0288</td>
<td>0.0160</td>
<td>0.0876</td>
<td>0.0396</td>
<td>0.0166</td>
</tr>
<tr>
<td>significance</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
<tr>
<td>$nsd$</td>
<td>0.01</td>
<td>0.25</td>
<td>0.5</td>
<td>0.75</td>
<td>1</td>
<td>1.25</td>
<td>1</td>
<td>1.25</td>
<td>1</td>
<td>1.25</td>
</tr>
<tr>
<td>$p$-value</td>
<td>0.0047</td>
<td>0.0178</td>
<td>0.0336</td>
<td>0.0290</td>
<td>0.0182</td>
<td>0.0047</td>
<td>0.0178</td>
<td>0.0336</td>
<td>0.0290</td>
<td>0.0182</td>
</tr>
<tr>
<td>significance</td>
<td>**</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td>*</td>
<td>*</td>
<td>*</td>
</tr>
</tbody>
</table>

Table 4.3: Statistical significance of the performance difference between OBN-CM and OBN-MSD. Two-tailed Mann-Whitney $U$ test: sample size $5 \cdot 10^4$; * = significance at $p < 0.05$, ** = significance at $p < 0.01$.

For these policies, the priority assignment heuristic appears to have a decisive influence on the performance and to outweigh the effects of the different stochastic estimation techniques. However, in general, SHEFT does perform slightly better than HEFT when using the same priority attribute, with the exception of SHEFT-T in the lowest processor pool size range, which suggests that the adoption of the MSD estimator in general does slightly improve the performance with respect to the MM (marginal mean) estimator. This is consistent with the results in the previous set, where the MSD estimator performed better than MM when applied to the OBN policy. In any case, it is also apparent that, with any of the available estimation methods, both HEFT and SHEFT would actually benefit from adopting the dominant sequence priority attribute, instead of the “default” alternative between b- and t-levels.

From the results in the fourth set, it is apparent that the AVG speed scaling strategy performs the worst for the expected response time under stochastic activity times, which has already been noted, for the makespan objective, by Zhao and Sakellariou [274]. The conclusion remains valid for the normalized response time. The two OBN algorithms (with offline partial moment-based and online conditional moment-based estimates) employing the MAX speed normalization method come close to the performance of the those adopting nominal speed method (NONORM) in many cases. However, MAX also behaves particularly badly under some parameter values: for example, for $p_{mul} = 0.9$ with the PM estimator and for $p_{mul} = 0.7$ with CM. MAX is also not robust in the high nsd range. Summarizing, it is the nominal speed strategy (NONORM) that generally delivers the best results. This may be explained, as in Sections 3.4, 3.6 with its properties with respect to (and its suitability in optimizing) the
makespan sub-objective.

Figure 4.6: Sample mean and dispersion of the response time for an instance, normalized by the processing time of all the task realizations in the instance’s critical path, as a function: (a) of the processor multiplier $p_{mul}$, given $|P| = ⌈vp_{mul}⌉$; (b) of the multiplier $\lambda_{mul}$ of the critical arrival rate, given $\lambda = \lambda_{mul}\lambda_{max}$. Algorithms with different job prioritizing and machine assignment heuristics.
Figure 4.7: (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation \( nsd = \frac{\sigma}{\mu} \) of the activity durations; (d) standard deviations of the normalized response time, as a function of \( nsd \). Algorithms with different job prioritizing and machine assignment heuristics.

In the fifth and final result set, the **OBN** policy outperforms its counterpart without
4.5. Experimental Evaluation

the active OBN delay mechanism (TOFF-NOOBN) and the reference greedy and lazy policies. Whereas the LAZY policy performs the worst, and is the least robust, the GDY policy does not fall much behind TOFF-NOOBN, except in the high arrival rate and normalized standard deviation ranges, where both trade-off-based algorithms are visibly better. On the other hand, the activation of the OBN delay mechanism seems to bring a decisive advantage in terms of the performance with respect to the expected response time objective. In detail, OBN delivers a performance better than TOFF-NOOBN consistently, although not by a large margin, except in the high nsd range, where the difference is more pronounced. Both trade-off-based policies are much more robust throughout the whole nsd range when compared to the plain lazy and greedy policies, with TOFF-NOOBN falling behind OBN for larger nsd values.

In figures, where the performance of the policies is measured against the size of the processor pool available in the scheduling platform, fluctuations may be observed. Given the experimental setup outlined at the beginning of this section, it is not guaranteed that the response time must be monotonically decreasing with a growing processor multiplier, or relative size of the processor pool with respect to the number of jobs in a single workflow instance, although this may be expected as a trend. Firstly, a larger number of processors does not imply a larger combined processing power (given that processor speeds are random) in a heterogeneous scheduling platform. Secondly, particular configurations with more processors may also be connected by actually slower channels. As both the proposed and the reference scheduling policies rely on the earliest finish time of a job (EFT heuristic) in the estimation and in the machine assignment phase, they do not take into account that, for a particular, fast processor, forwarding the outputs may be effectively slower than elsewhere. These effects have not been explicitly counteracted, because they in fact may occur in many practical settings and realistic heterogeneous platforms and the evaluation has been aimed at determining the comparatively best algorithm choices.

Summarizing, the adoption of the OBN delay mechanism significantly improves the proposed policy in terms of the response time. For the same objective, the unnormalized speed scaling strategy is visibly better than any other, previously adopted in scheduling on heterogeneous platforms. The conditional partial moment-based estimators in most cases deliver a better performance and robustness than the partial moment-based ones or the reference MM and MSD estimators. Finally, the trade-off
based variable greediness delivers better performance than plain lazy or greedy algorithms, idle time charging or fixed priority-driven greediness (HEFT, SHEFT, DLS).

Figure 4.8: Sample mean and dispersion of the response time for an instance, normalized by the processing time of all the task realizations in the instance’s critical path, as a function: (a) of the processor multiplier $p_{mul}$, given $|P| = \lceil vp_{mul} \rceil$; (b) of the multiplier $\lambda_{mul}$ of the critical arrival rate, given $\lambda = \lambda_{mul} \lambda_{max}$. Algorithms with different processor and channel speed normalization strategies.
4.5. Experimental Evaluation

Figure 4.9: (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $nsd = \frac{\sigma}{\mu}$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $nsd$. Algorithms with different processor and channel speed normalization strategies.
Figure 4.10: Sample mean and dispersion of the response time for an instance, normalized by the processing time of all the task realizations in the instance’s critical path, as a function: (a) of the processor multiplier $p_{mul}$, given $|P| = \lceil vp_{mul} \rceil$; (b) of the multiplier $\lambda_{mul}$ of the critical arrival rate, given $\lambda = \lambda_{mul} \lambda_{max}$. Algorithms with different cross-instance strategies.
Figure 4.11: (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $nsd = \frac{\sigma}{\mu}$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $nsd$. Algorithms with different cross-instance strategies.
Table 4.4: Statistical significance of the performance difference between OBN and TOFF-NOOBN. Two-tailed Mann-Whitney $U$ test: sample size $5 \cdot 10^4$; * = significance at $p < 0.05$, ** = significance at $p < 0.01$, *** = significance at $p < 0.001$.

4.6 Summary

In this chapter, a general scheduling policy for the the $G/G/c : P/Q | v_i, c_{ij} \sim \text{stoch}$, $prec | E[W]$ has been proposed. This is a problem that focuses on the minimization of the expected response time, when a queue of directed acyclic graph-shaped workflow instances is scheduled onto either a homogeneous or uniformly heterogeneous platform. Both the activities within the workflow and the instance arrival process are allowed to be described by arbitrary stochastic variables, where the respective distributions are only subject to very mild assumptions. In particular, the distributions of the activities must be known with respect to their cumulative distribution functions and must have finite first and second partial moments in the sense of Sect. 4.4.1.

The proposed policy is applicable online, i.e., during the scheduling process, in a fashion that follows the pattern of elementary policies (see [2.19]), given its relatively low time-complexity. It employs novel, conditional partial moment-based estimates that allow for an accurate scalar description of stochastic variables for the purposes of a list scheduling algorithm. It also takes advantage of the accuracy of such estimates by making predictions on the start and finish times of high and low priority jobs. In this manner, it is able to determine to which extent a low priority job is likely to delay the execution of a high priority one to be scheduled in future. It can thus decide whether the scheduler should execute the low priority job greedily or wait for the high priority one to become ready. The policy also includes an optional local look-ahead subroutine specifically targeted at minimizing the waiting time of new workflow instances by al-
lowing them to start before the termination of the instances currently under processing, when the termination of the instances of higher priority is unlikely to be affected. The proposed scheduling policy was examined in a variety of experimental settings against state-of-the-art algorithms and several variations of other comparable schedulers. All of its component heuristics were also tested separately against possible alternatives. With respect to both performance and robustness, the scheduling policy being the contribution of this chapter outperforms the competitors, demonstrating the suitability of its design choices. The difference is more pronounced in settings with medium processor pool sizes and when the uncertainty of the inputs (the normalized standard deviation of the activity runtimes) is not excessive. The conditional partial moment-based estimators proposed in this chapter bring the most advantage especially for low relative sizes of the processor pool and higher relative arrival rates, i.e., when the total number of jobs per time unit is large with respect to the total processing capacity of the system. The out-branch-node (OBN) delay mechanism significantly improves the expected response time in all experimental settings when applied to the trade-off-based scheduler.
Chapter 5

Scheduling of Petri Net-reducible Stochastic Workflows

The previous chapters investigated the problem of scheduling a single instance or multiple instances of a workflow, whose application model is expressed by a directed acyclic graph (DAG). However, many realistic scenarios require an application model able to express a broader range of control flow constructs. In particular, actual workflows feature conditions evaluated at runtime: Based on the result of the evaluation of such conditions, choices are made that affect the set and the order of activities to be executed. One or more of the alternative paths originating in a decision point (split) can be chosen or a subset of the activities defined in the workflow may be repeated. An exclusive choice (XOR-split), trivially, corresponds to the constraint that only one of the alternative branches will be executed. Conversely, a non-exclusive choice (OR-split) determines any number of branches originating in the split to be executed in parallel. These features are termed conditional control flow and iterative control flow, respectively; in iterative flows, the set of activities that may be repeated is called a loop.

This chapter investigates a range of problems that arise when such extensions to the ordinary series-parallel control flow expressible in DAGs are allowed.

The first problem is an appropriate choice of application model, which, in addition to the expressive power required, must allow for efficient scheduling solutions. In our approach, the application model is assumed to be an extension of a workflow net. By itself, a workflow net is a Petri net subclass that represents a syntactically correct
workflow schema: It is a bipartite graph with two sets of objects, transitions and places, connected by a flow relation (a set of directed edges). In the broad context of workflow management, the transitions are associated with workflow tasks and the places represent pre- and postconditions of tasks, or evaluation of conditions. Here, we further assume that a subset of the edges represent a communication delay (data transfer) between connected jobs. This communication is required if these tasks are assigned to different processors, otherwise it is optional. Both the job runtimes and the communication delays are assumed to be described by stochastic variables, with arbitrary probability distributions having a known cumulative density function. A place with multiple outgoing edges to transitions represents a choice of alternative flows: We additionally assume that any such place is associated with a probability density function describing the likelihood of each choice.

The second problem is the specific scheduling problem of minimizing the total time spent in the system by an average workflow instance conforming to the chosen application model, as perceived by the end user, i.e., the expected response time. The workflow instances are submitted into the system by a stochastic arrival process that gives rise to a queue. The arrival process is not required to follow any specific distribution. The scheduling platforms examined are homogeneous and heterogeneous computing environments, with a known number of identical or uniform processors, respectively, fully connected by identical or uniform physical channels (see 2.2.2).

As in the previous chapter, it is assumed here that real-time dispatching of tasks is allowed. This makes it possible to use an online approach to both the determination of task priorities and to the machine assignment phase. Therefore, it is possible to update the job priority lists dynamically, based on the actual activity execution times. Moreover, runtime monitoring allows the scheduler to update the list of potentially schedulable jobs by discarding conditional branches and loops that are no longer relevant for a particular workflow instance. The stochastic estimates of the remaining conditional branches and loops can thus be recomputed and the updated partial path length estimates can be used for a more accurate job priority determination. This includes recomputing, when applicable, the remaining length of paths, including loops, which contain activities that are currently running, i.e., are executing when the policy is to make scheduling decisions.

A baseline first-in, first-out (FIFO) queue discipline governing the processing of work-
flow instances is assumed. However, in many cases it is possible to execute jobs belonging to more than one instance at a given time. In particular, this may happen when it is unlikely that the makespan of an instance of higher priority may be affected by a decision to schedule a set of jobs belonging to one of lower priority on some of the idle processors.

The following contributions are put forward:

1. A scheduling policy for the problem of minimization of the expected response time, when a queue of instances of a workflow net with stochastic activities is scheduled onto a homogeneous or uniformly heterogeneous platform, i.e., the $P/Q\mid v_i, c_{ij} \sim stoch, wn \mid E[W]$ problem (see 2.12); within a generic online scheduling framework, it enables the employment of different conditional path and loop length estimation strategies, in addition to the different stochastic variable value estimation methods and job priority determination heuristics supported by the policies proposed in the previous chapters,

2. A computationally efficient method of determining the set of executable jobs, based on labelling of inactive jobs and propagated elimination of jobs whose parents are labelled inactive,

3. An efficient method of estimating the length of conditional paths involving stochastic activities, which supports both exclusive and non-exclusive choices,

4. An efficient method of estimating the length of loops containing stochastic activities, which supports arbitrary combinations of loops with a finite or potentially infinite number of iterations and provides more accurate estimation methods for loops containing currently running activities,

5. An evaluation of the schedules derived with the policy proposed and of the alternative solutions with respect to their cross-instance robustness (see 2.17).

5.1 Motivating Scenario

Consider the scenario from Figures 5.1 and 5.2. Each of the figures shows a version of the workflow corresponding to the skilled migrant visa assessment process in the De-
5.1. Motivating Scenario

The Department of Immigration and Citizenship (DIAC) of the Commonwealth of Australia. This example, already featured in a simplified form in Sections 3.1 and 4.1, is adapted from Wynn et al. [269]. The workflow is modelled with the YAWL editor: for YAWL (Yet Another Workflow Language [230]) see Sect. 5.2.

![Figure 5.1: A possible DIAC skilled migrant visa processing workflow with a conditional control flow [269]. The circles represent conditions evaluated during execution, based on the output of the preceding tasks. The evaluation output (“pass”/“fail”) relates to the reported example, but in general it is not required to be binary in the model (see also Figure 5.2) nor in the scheduling algorithms proposed hereafter.](image)

The first figure and version of the scenario still presents a slight simplification, by which only a set of choices are included as conditional branches. These are associated with the evaluation of the case documents, including language skills, core skills and health checks. The second is complete with an option of requesting additional information of the applicant, which must be followed by a re-evaluation of the visa application, including a recount of the relevant skill points. This gives rise to a potential loop in the process, which may be executed, theoretically, any number of times.
As in the previous chapter, we assume that all of the employees in the department have passed the necessary training and obtained the clearance to perform any of the tasks associated with the visa application. Different employees may process different kinds of tasks at different speeds. In addition, a particular client’s case (i.e., workflow instance) will incur delays associated with the transfer of certain case information, if different employees execute the tasks within the case: for example, when working copies of the client’s files are sent to the next case officer. Thus, in this scenario, a communication delay is only induced when an actual transfer of data between two separate processing units is required.

5.2 Suitability of Workflow Models

In the previous chapters of this thesis the directed acyclic graph (DAG) is assumed as the underlying application model. In many scheduling approaches known from the literature, the employment of the DAG model often originates from a reduction of the complexity of a more sophisticated original application model. For example, Kwok and Ahmad [143] presented an argument applicable to the dataflow domain, which concerns itself with problems such as Gaussian elimination (GE) and Fast Fourier Transform (FFT) computation algorithms. They argued that, in many classes
5.2. Suitability of Workflow Models

of dataflow problems, there are very few conditional branches, or none, as in matrix
multiplication. Similarly, in dataflow problems where loops are present, it is usually
possible to execute these loops in parallel, when their input data become available.
Two variants of a technique based on this principle, called “loop unravelling”, were
proposed by Beck, Pingali and Nicolau [25] and Lee, Hurson and Feng [148]. Kwok
and Ahmad [143] also argued that, in many typical dataflow applications, such as GE
and FFT, the bounds on the length of a loop can be strictly determined at compile time,
which makes it possible to virtually encapsulate a single loop execution into a single
DAG task node. In connection with loop unravelling, multiple loop executions can be
statically planned and represented by a single virtual task.

A substantial body of research in the field of compiler design applies loop transfor-
mations [5, 22, 47]. The so-called software pipelining [144] speculatively rearranges
the order of other tasks outside of the loop, in its vicinity. An example of this is
Rau’s iterative modulo scheduling framework [197]. The number of unfolded loops
is adjusted with respect to the available processing resources in more advanced algo-
rithms [65, 159]. Most of these approaches rely on the assumption of deterministic
processing times for in-loop tasks. An exception is the work of Tongsima et al. [232],
who introduced two heuristics, probabilistic retiming (PR) and probabilistic rotation
scheduling (PRS) for problems with an unbounded or bounded number of processing
units, respectively. Both take as an input a probabilistic dataflow graph (PG), which is
a directed graph of task nodes with a small number of cycles, and a target confidence
interval for the makespan of the schedule.

The reduction techniques presented above and their motivation are specific to the do-
mains for which they were originally postulated. In particular, the loop unravelling
method requires out-tree-shaped task graphs, i.e., graphs where a fork in the execution
path is not followed by a join, or synchronization. When the number of loop execu-
tions is not known a priori, it is also not convenient to allocate jobs belonging to loops
that may not be executed at all, especially in online scheduling contexts, where reac-
tive policies may be applied. Moreover, the assumption that only very few workflow
models of interest may have conditional branches is not justified beyond the dataflow
domain as defined in Kwok and Ahmad’s work [143]. The previous section of this
chapter has presented example scenarios that motivate the necessity of considering,
for realistic office workflow execution problems, a broader range of control flow con-
structs than possible using the DAG model. The reduction and schema simplification techniques outlined above are clearly not suited for these scenarios.

To capture these requirements, an application model is needed that explicitly supports conditional control flow and unbounded, possibly inter-dependent iterations (loops), in addition to all the series-parallel flow combinations possible in arbitrary DAGs. At the same time, it is desirable that the model be syntactically similar to the DAG model and close to its minimal superset having the required expressive power: overly complex models should be avoided. Finally, it should be formally well defined and have analytically determinable properties.

Many widely used modern workflow modeling and execution languages originated in the business management domain, with the purpose of providing a structured and reusable framework for office activity, service and production design, organization and execution. The most notable examples include Unified Modeling Language (UML) activity diagrams (see Dumas and ter Hofstede [63]), Business Process Model and Notation (BPMN, White [259]) constructs and Event-driven Process Chain flowcharts (van der Aalst [245]). These methodologies and corresponding tools extended, largely reworked and finally superseded earlier approaches such as the classical Gantt charts or the Program Evaluation and Review Technique (PERT) and many others, developed before standardization efforts begun in the 1990s. The objective of these standardization efforts in the business domain, however, was mainly reducing the number of competing specification languages, not minimal syntax or formal rigour within a single language. As a result, all of these approaches make use of partially overlapping constructs ([251], [265]), many of which only have intuitive semantics [268]. This gives rise to contradictory interpretations even for the basic control flow constructs [135]. A correlated issue is the fact that they are not amenable to formal analysis: The need for the research community to address this was individuated quite early, specifically in the works of Georgakopoulos, Hornick and Sheth [82] and Alonso et al. [11]. This motivated a number of formally well-founded approaches, including different flavors of logic [174].

A substantial body of research work advocates the use of Petri nets for this purpose ([117], [244], [246], [210]), mainly because of their formal nature, the availability of analysis techniques and the fact that their state-based approach can naturally model state-based scenarios [242]. An additional advantage is their intuitive graphic repre-
5.2. Suitability of Workflow Models

Presentation [182]. More importantly, Petri nets can express control flow constructs such as sequence, parallel and exclusive splits and joins and iteration ([117], [241]), including those that are impossible to formulate in state-of-the-art process algebras [248]. In fact, they can model all of the control flow patterns individuated by research ([250], [203]), with the exception of specialized constructs such as cancellation or synchronization which involve non-local semantics or multiple instances ([244], [229]).

The scenarios considered in this thesis do not feature any of the latter patterns, thus Petri nets are assumed in the following as the language of choice for the application model (workflow schema). An additional advantage of the employment of this model is the availability of many business process design tools (e.g., ProM [252], OWLS2PNML and others) that can import and export process models in an XML-based interchange format, the Petri Net Markup Language (PNML, [30]).

The additional requirements posed by the above mentioned non-standard control flow patterns led to the development of Yet Another Workflow Language (YAWL, [230]). It supports both cancellation, i.e., preventing or interrupting the processing of a subset of the workflow’s task based on a task’s outcome, OR-joins with multiple optional branches, interaction of multiple instances and hierarchical process design. YAWL also offers an intuitive, synthetic graphical workflow modeling language, which is used throughout this thesis for a high-level representation of the scenarios of interest (e.g., Figure 3.1, 4.1, 5.1). For each of these examples, the equivalent Petri net would be much larger and far less intuitive.

5.2.1 Petri Nets and Workflow Nets

First proposed by Petri in his thesis [186], Petri nets [185] were originally a model of control flow in a concurrent system. They have been applied in a large number of analysis approaches and tools, ranging from offline model checking (see, e.g., Berthomieu and Diaz [28]) to online simulation (Nou et al. [180], Yoon and Shen [271], ElMaraghy et al. [68]).

The following definitions are reported in a formalization largely due to Wynn [268].

**Definition 5.1.** A Petri net, or place-transition net, or P/T-net, is a tuple \((P, T, F)\) where \(P = \{p\} \neq \emptyset\) is the set of places, \(T = \{t\}\) with \(T \cap P = \emptyset\) is the set of
transitions and \( F \subseteq (P \times T) \cup (T \times P) = \{ f : f = (p, t) \lor f = (t, p) \} \) is the set of arcs (the alternating flow relation) between a subset of the members of \( P \) and of \( T \).

Given \( x \in P \cup T \), the preset \( \bullet x \) of \( x \) is the set of objects \( Y_{\text{pre}} \) (places, if \( x \in T \), or transitions otherwise) in the Petri net for which it holds \( Y_{\text{pre}} = \{ y : (y, x) \in F \} \). Conversely, the postset \( x \bullet \) of \( x \) is the set of objects \( Y_{\text{post}} \) for which it holds \( Y_{\text{post}} = \{ y : (x, y) \in F \} \).

In the workflow context, transitions usually represent tasks (atomic jobs) and places their pre- and postconditions \([244]\). We shall additionally assume in the following that inter-task communications are mapped to the place-transition arcs and that each activity is annotated with a reference to a real-valued variable describing its duration.

A Petri net can be also regarded as a bipartite graph. It is also usually graphically represented as such, with transitions (jobs) indicated by bars or rectangles, places represented by circles and arrow-shaped arcs (Figure 5.3). The postset of a place is the set of its output transitions, as indicated by the directed arcs. A similar reasoning can be applied to any transition as the starting point and to presets (with arrows followed “backwards”). In the workflow context, the postset of a transition \( t \) represents the set of postconditions that are true after the completion of the job corresponding to \( t \). Conversely, the preset is the set of preconditions which must be true for \( t \) to begin execution. The actual occurrence of pre- and postconditions during the execution of a workflow instance is indicated by the presence of tokens, or marks, in the respective places in the net. This is the basic execution semantics provided by Petri nets and formally based on the following definitions.

**Definition 5.2.** Given a Petri net \((P, T, F)\), a marking is a function \( M : P \to \mathbb{N}^n \) with \( n \in \mathbb{N}_+ \). For a place \( p \) in the net, the place marking function \( M(p, k) \), with \( M(p, k) \in \mathbb{N} \), conventionally describes the number \( n \) of marks (tokens) of type \( k \) in \( p \).

In the broad context of Petri net applications, the interpretation of the tokens is application-driven and fairly arbitrary. For example, tokens of different types may represent various kinds of resources needed for the execution of the process modelled by the Petri net, or products and by-products of such an execution. To simply model the control flow of a concurrent system, a single type of token is sufficient. In that case, the tokens just indicate the occurrence of preconditions and postconditions of an event, as outlined above.
5.2. Suitability of Workflow Models

**Definition 5.3.** A Petri net is ordinary if it admits only one type of tokens, i.e., requires \( k = 1 \). Thus, the place marking function of an ordinary Petri net can be expressed by \( M(p) \).

An ordinary Petri net is 1-safe if the number of tokens in any place in the net never exceeds one, i.e., \( \forall p \in P \Rightarrow M(p) \leq 1 \).

**Definition 5.4.** Given an ordinary Petri net \( (P, T, F) = N \), a transition \( t \) is enabled if each place in its preset contains at least one token, i.e., if \( \forall p \in \bullet t \Rightarrow M(p) \geq 1 \).

A transition that is enabled may fire. Firing is an atomic act (step) which consumes a single token from each of the places in the preset and generates exactly one token in all of the postset places. This is written \( M \xrightarrow{t} M' \).

A firing sequence is any feasible order of transitions \( \sigma(M, N) = \{t_1, \ldots, t_N\}, < \) that may fire in \( N \) steps given a marking \( M \).

A marking \( M' \) is reachable from a marking \( M \) when there exists a firing sequence \( \sigma(M) \), of any length, from \( M \) to \( M' \), written \( M \xrightarrow{\sigma(M)} M' \).

The set of all markings that are reachable from \( M' \) is indicated as \( N[M >] \).

The fact that a transition \( t \) is enabled by a marking \( M \) is written \( M[t >] \).

A transition that would be enabled by some reachable marking is called potentially fireable, or live (dead otherwise).

In ordinary Petri nets, the marking is typically interpreted as the state of execution of the modelled system (following Petri’s original work). This interpretation is assumed also throughout this chapter. Strictly speaking, the marking as a whole corresponds to the state of the control flow process associated with the executing workflow instance. In particular, a token in an output place of a transition (any of the places in its postset not contained in the postset of another transition) signifies the fact that the task associated with that transition has been completed. A token in each of the preset places of a transition indicates that the precedence constraints for the corresponding task are satisfied. When a transition is fired, it instantly removes (consumes) a token from each of its preset places and, when it finishes executing, it instantly puts (creates) a token in each of its output places: Tokens are not “moved”, as there may be more or fewer preset than postset places. Additionally, it is possible for a transition to be multiply connected to the same postset place and thus to produce multiple tokens in that place.
when firing. The evolution of the state of the control flow as described by the consumption and creation of tokens is termed token game. With no tokens, the unmarked net corresponds to the workflow model which is shared by all the instances.

The reachability of a marking and the liveness of transitions are $\text{EXPSPACE}$-hard problems in ordinary Petri nets, as shown by Lipton [157], and $\text{PSPACE}$-hard in 1-safe ones, as proved by Jones, Landweber and Lien [121]. These complexity classes are related by the following relationships: $\text{NP} \subseteq \text{PSPACE} \subset \text{EXPSPACE}$, see, e.g., Rich [198]. A survey of complexity results in Petri nets is given by Esparza [72], while Praveen’s work [192] includes specific results for restricted Petri net classes.

Certain structural properties of Petri nets are of particular importance for the problems discussed in this chapter. These properties are formulated based on the auxiliary definitions 5.5, 5.6 which follow.

**Definition 5.5.** A Petri net is conflict-free if, for any of its places, the size of the postset is not larger than one.

In a non-conflict free Petri net a preset place can be shared by two or more transitions. Given a single token in this place, the transitions are in conflict, as they compete for the token and only one of them may fire. Firing one of the conflicting transitions models a choice. In other words, a token in a place which has multiple transitions in its postset enables an OR-split. This split may be exclusive (OR) or non-exclusive (XOR), depending on how tokens are produced and consumed. In 1-safe Petri nets the choice is always exclusive, as there may be only one token to consume. Conversely, in 1-safe nets, a token in a place in which only one of two or more transitions can produce a token enables an exclusive OR-join. Strictly speaking, the act of joining is associated with the firing of the output transition(s) of such a place.

Non-exclusive choices are more conceptually complex and require an extended execution semantics. Given $n$ paths that can be enabled at the split, any $m$ paths can be chosen, with $1 \leq m \leq n$, while the others must remain inactive. Potentially, $m$ parallel threads can be started. Another formulation of this fact is that, in a concrete workflow instance, the OR-split gives rise to an AND-split (one of the possible combinations of AND-splits). Additionally, if there exists a reachable place $p_j$ with multiple input transitions, where the paths split previously intersect, different behavior patterns are possible. In particular, the execution of the workflow instance may continue immedi-
5.2. Suitability of Workflow Models

ately after the fastest execution thread reaches place \( p_j \) (produces a token there), while the work performed by the other threads is discarded. In that case, no thread synchronization takes place. In other words, an XOR-join effectively follows an AND-split. This pattern is termed simple merge (van der Aalst et al. [250]) or single merge (Zheng et al. [276]). Alternatively, the execution of the workflow instance may continue only after all the threads started at the split reach place \( p_j \) (synchronizing or synchronized merge [250], [276]).

To model non-exclusive choices it is not sufficient to allow multiple tokens in a place with multiple output transitions. Firstly, a total of \( n \) tokens in a place can enable \( n \) output transitions simultaneously, but if only \( m < n \) transitions are required to fire, the remaining tokens must be destroyed by some mechanism external to the ordinary Petri net execution semantics. Moreover, without additional assumptions any of the transitions could fire multiple times. Finally, when multiple execution threads after a split are allowed to produce tokens in a place \( p_j \) where the corresponding paths join in the net, only a simple merge can be modeled: The first token produced in \( p_j \) already enables further execution. To enforce thread synchronization in a place \( p_j \), an execution monitor must keep the information on which of the transitions in the preset of \( p_j \) will eventually be enabled in a particular workflow instance. This knowledge cannot be derived solely by observing the token production process in \( p_j \). This problem is referred to as non-local semantics of OR-joins in the literature ([250], [268]). In other words, the monitor must know which one of the possible AND-joins must be enforced in a given workflow instance, such that this join corresponds to the actual AND-split that precedes it.

We assume that an execution monitor is available as part of the scheduling algorithms considered in this chapter. In this manner, the formulation of an execution semantics is possible which enables modeling of non-exclusive choices and synchronizing merges. In particular, in the extended semantics proposed, given a token in any place which has multiple output transitions, more than one transition can fire. At runtime, the set of the transitions actually fired is known to the execution monitor, which can thus manage thread synchronization, if required. The synchronization mechanism is responsible for producing a token in the place in which the paths corresponding to the threads intersect.

**Definition 5.6.** In a Petri net \( (P, T, F) \), let the transitive closure of the flow relation \( F \) be indicated as \( F^+ \). A structural loop in the net is the set \( \Lambda \) of all objects \( x, y \in P \cup T \),
$(x, z) \in F$ such that:

1. there exists a path from $x$ to $y$ and vice versa: $(x, y), (y, x) \in F^+ \Rightarrow x, y \in \Lambda$

2. the arcs on a path satisfying point 1. belong to the loop: $\forall (x, z) \in F | (z, x) \in F^+ \Rightarrow (x, z) \in \Lambda$

3. the paths from $x$ to $y$ and vice versa are unique: $\forall z_i, z_j \in y \bullet (z_i, y), (z_j, y) \in F^+ \Rightarrow z_i = z_j$.

A loop entry point is any place or transition belonging to a loop which has an object (transition or place, respectively) not belonging to the loop in its preset.

A loop exit point is any place or transition belonging to a loop which has an object (transition or place, respectively) not belonging to the loop in its postset.

A Petri net with no structural loops and no OR-splits and joins can be mapped to a directed acyclic graph. This may be done by generating a DAG node for each transition in the net and connecting the graph’s nodes directly in accordance to the original flow relation, while discarding the net’s places. Formally, given such a restricted Petri net $(P, T, F)$, the corresponding DAG is a tuple $(V, E)$, with $V = T = \{t_i\}$ and $E = \{e_{ij}\}$, where:

$$\forall f_i = (t_i, p), f_j = (p, t_j) \in F \Rightarrow (f_i, f_j) \mapsto e_{ij} = (t_i, t_j) \quad (5.1)$$

Thus, the Petri net model not only provides the minimal required superset of features in terms of expressive power with respect to the DAG model but also a trivial mapping to it. It is easy to see that this mapping is bi-directional.

The previous discussion concerns Petri nets with no structural restrictions. However, a body of work dedicated to workflow modeling and verification issues led to formulating additional restrictions with which verifiable, well-behaved workflow models must comply. The workflow net (WF-net), first proposed by van der Aalst [244] and revised by Verbeek [254], is such a model. Informally, it represents an application with a correct control flow structure, with clear start and termination points, and a guaranteed sound execution [249]. Correctness means that there are no unreachable, “dangling” parts of the net and soundness implies that proper termination is possible and there are no dead transitions. Formal definitions are provided below.
5.2. Suitability of Workflow Models

Definition 5.7. A workflow net is a Petri net with exactly one start place \( p_s \), exactly one end (sink) place \( p_e \) and such that all the places and transitions are on a path from \( p_s \) to \( p_e \), i.e., a Petri net that satisfies the following conditions:

1. \( P_s = \{ p \in P \mid o_p = \emptyset \} \Rightarrow |P_s| = 1 \wedge p_s \in P_s \)
2. \( P_e = \{ p \in P \mid p \bullet = \emptyset \} \Rightarrow |P_e| = 1 \wedge p_e \in P_e \)
3. \( \forall x \in P \cup T \Rightarrow (p_s, x) \in F^* \wedge (x, p_e) \in F^* \)

where \( F^* \) is the reflexive transitive closure of the flow relation \( F \).

Definition 5.8. Let \( M_s \) be a marking with a single token in \( p_s \) and no tokens elsewhere and let \( M_e \) be a marking defined analogously for \( p_e \). A workflow net is sound if it satisfies the following conditions:

1. option to complete, i.e., \( M_e \) can be reached, formally: \( M_e \in N[M_s] > \)
2. proper completion, i.e., \( M_e \) is the only marking with a token in \( p_e \) that can be reached: \( \forall M \in N[M_s] > \mid M(p_e) = 1 \Rightarrow M = M_e \)
3. no dead transitions: \( \forall t \in T \Rightarrow \exists M \in N[M_s] > \mid M[t] >. \)

The following can be observed. In a workflow net, for each split of two paths there must be a corresponding join and vice versa, as there is exactly one start place and end place. Moreover, the corresponding join must be of the same kind as the split. Trivially, an AND-join of any two paths originating in an XOR-split would cause a deadlock (no option to complete). On the other hand, an XOR-join of any two paths originating in an AND-split always results in improper completion, as the execution of the workflow can continue with a token left in the place where the paths join. Notably, in reference to the splits and their correspondence, it is always possible to partition the workflow net into sequences, loops and parallel branches which can execute concurrently (AND-branches) or alternatively (XOR-branches). Thus the workflow net model can be viewed as a structured model. Workflows without this property are often encountered in practice, also because fewer restrictions allow for more creativity in the model design (Hopkins [107]). They were investigated both in the domain of Petri nets and event process chains (see, e.g., Polyvyanyy [191]). However, as it follows
from the observations above, in unstructured Petri nets deadlocks are possible or im-
proper completion is guaranteed, or both. The approach presented in this chapter only
supports workflow nets and thus structured workflows.

Figure 5.3: A Petri net with a single start place $p_s$ and a single end (sink) place $p_e$.
Without the two dotted arcs $(p_3, t_1)$, $(t_4, p_2)$ the net is also a correct workflow net in
the sense of Def. 5.7. A single token in $p_2$ enables both transition $t_2$ and $t_3$, but only
one of them may fire. Thus $p_2$ corresponds to an XOR-split: $p_5$ is the related XOR-
join. The branches following $p_1$, $p_2$, $p_3$ are AND-parallel branches, i.e., correspond to
parallel branches in a DAG. Transitions $t_4$, $t_5$ are in a loop. The dotted arc $(t_4, p_2)$
makes the net not 1-safe (Def. 5.2), while $(p_3, t_1)$ may cause a deadlock.

5.3 Problem Formulation

Let $W = (P, T, F)$ be a sound workflow net. Let $J$ be the subset of $T$ corresponding
to the set of all schedulable jobs. All members of $J$ have a nominal processing time,
given by a stochastic variable $w(j_i)$ with a known arbitrary probability distribution,
whose first and second partial moments and partial conditional moments are defined
and finite (see 3.5.1 and 4.4.1). Let $E$ be a subset of $F$ corresponding to the set of inter-task
communication links. Each of the members of $E$ is associated with a nominal transfer
delay between two directly connected jobs $j_i, j_k$. In the workflow net, two jobs are
directly connected if they share a connection that includes two members of $E$ and no
other member of $J$. The delay is given by a stochastic variable $w(e_{ik})$, satisfying the
same requirements as $w(j_i)$. Strictly speaking, we associate the members of $E$ to a
subset of those members of $F$ which are incoming arcs defined for members of $J$. An actual delay due to communication, however, occurs only when the two directly connected jobs are scheduled to two different processors.

Let $p$ be a preset place of the transition $t_k$ corresponding to job $j_k$. In 1-safe Petri nets, if the size of the postset $p^\bullet$ of $p$ is larger than one, then $p$ corresponds to an XOR-split. In terms of execution semantics, we assume the following extension of the ordinary Petri net token game. A single token in place $p$ may cause any non-zero number of transitions in the postset of $p$ to fire. Let the marginal probability of choosing any transition $t \in p^\bullet$ after $p$ be given by a stochastic variable with an arbitrary known distribution. Let additionally the event of choosing at least one transition be certain. On the other hand, if $p$ is an exit point of a loop, we assume that the probabilities of choosing any transition in its postset add up to one in any loop iteration, i.e., the split in $p$ is exclusive-OR (XOR).

Let there be a stochastic process describing the arrival of new workflow instances into the system. No assumptions are made with respect to its distribution. For some specific parametrizations of the proposed scheduling policies, it is required that the average arrival rate $\lambda$ be known.

The scheduling platform consists of $Q$ with $|Q| < \infty$ parallel processors, fully connected by a set of $H = Q \times Q$ communication channels. The platform may be homogeneous or uniformly heterogeneous (see Sect. 2.2.2).

The $G/G/c : Q|v_i, c_{ij} \sim sto\text{ch}, wn \ E[W]$ problem can be formulated as follows: minimize the expected total response time $E[W]$, over an infinite horizon (number of instances), when a queue of instances of the known workflow net is scheduled to the platform defined by $(Q, H)$.

## 5.4 Related Work in Scheduling

As summarized by Kwok and Ahmad [143], the properties of task graphs usually found in parallel processing problems in the dataflow domain allow for simplifying assumptions. In particular, conditional branches can be ignored on the basis of their rare occurrence and the length of the loops can be estimated statically. Thus, a reduction of these graphs to directed acyclic graphs is possible. This is a clear advantage especially
in the context of offline scheduling, where runtime information on which conditional branches and loops should be actually executed is unavailable. Clearly, an offline scheduling policy applied to a task graph with irreducible conditional branches and loops must explore a much larger decision space than it is the case in DAG scheduling or online scheduling problems.

There are, however, examples of offline policies supporting conditional or iterative control flow constructs. Chou and Abraham [51] proposed a computational model for distributed programs which included conditional branching but no loops. They represented the task assignment problem in a parallel computing environment as a Markov decision process, which models probabilistic branches in a natural way, and solved it by using a policy iteration algorithm. Motivated by the fact that an MDP-based scheduler is inherently inefficient (see Sect. 2.11), Towsley [235] adapted Bokhari’s shortest path algorithm instead [37] to the case of graphs with probabilistic branches and loops. Towsley’s solution, however, only supported problems with a limited number of loop executions, known a priori. In addition, as the original algorithm was designed for tree-like task graphs, it was obviously sub-optimal for other types of precedence constraints. Price and Pooch [193] also applied a shortest path algorithm to the same problem, however, in a fundamentally different manner. They designed a multi-stage approach: the original problem was first formulated as a zero-one quadratic programming one, which could be represented by a directed acyclic search graph. Subsequently, the shortest path algorithm was employed to produce a coarsely optimized task-to-processor assignment from this graph. To refine this assignment, a non-backtracking branch-and-bound local search method was used.

El-Rewini and Ali [66] proposed another multi-stage method for static scheduling problems with conditional branches. They split the application model into two graphs: the familiar precedence graph and a branch graph (which is also a DAG). The first stage of their approach applied a set of graph-theoretic reductions to both graphs in an attempt to reduce the search space for the second phase, which entailed the actual scheduling process. The reduction was mainly based on similarity between tasks in different conditional branches. This, however, poses a fairly strict requirement: the alternative tasks considered must have the same input and output tasks. The remaining irreducible alternatives were used to generate a set of DAG task graphs representing each a complete possible execution of the application. Each such graph was assigned
5.4. Related Work in Scheduling

An overall execution probability, obtained by multiplying the probabilities of each conditional branch inherited from the original branch graph. Thus, the number of DAGs was exponential in the worst case. A simple mapping heuristic scheduled a subset of these alternative graphs simultaneously in a manner expected to reflect their frequency. If the number of alternative graphs exceeded a threshold, only the most likely ones were selected for scheduling until all of the atomic tasks in the application were considered at least once. The final schedule resulted from merging the schedules of the selected graphs. The authors claimed their approach considered loops, arguing that the indeterminism associated with other forms of conditional branching, for which they provided a solution, was larger. However, this claim is only valid for loops that are reducible to a single virtual task, i.e., mainly for loops with a trivial internal structure and a known number of iterations.

Santoshkumar, Manimaran and Murthy [211] applied the same principle for the problem of scheduling in the presence of deadlines. The alternative DAG task graphs (variants) derived from a single application model with conditional branches were given weights based on two elementary strategies or a mix of them. The first strategy assigned weights proportionally to the execution frequency of the variant or its assumed likelihood, as in [66]. Naturally, larger weights corresponded to a higher priority. The second strategy identified, for each variant, the length of its critical path as the weight. The final schedule was constructed by merging the partial schedules while ignoring variants of lower priority. Loops were not explicitly supported. Note that the approaches in [66], [211] are also similar in applicability as they are strictly dependent on deterministic activity times in the application model.

A similar conditional task model, a superposition of DAGs with assigned probabilities, was investigated by Huang and Oudshoorn [110]. They derived it by means of an explicit process mining subroutine, that also updated the expected jobs’ computation times. They proposed a simple scheduler called CET, which is a greedy earliest start time-based heuristic for all the “obligatory” jobs, i.e, those that must be executed in at least one of the DAGs. The remaining, “optional” jobs are scheduled giving priority to the ones with the smallest co-level, which is sum of the t-level and the expected execution time of a task, with no regard to their specific probabilities.

Jakoby, Liśkiewicz and Reischuk [118] proposed a dynamic process graph (DPG), a model which allows for a compact representation of task graphs with conditional
branches and communication delays. They were able to prove that makespan mini-
mization with communication delays, which is \( \text{NP} \)-complete in the case of arbitrary 
DAGs, becomes \( \text{NEXPTIME} \)-complete in DPGs, i.e., is strictly more complex, given 
that \( \text{NEXPTIME} \supset \text{NP} \) [98].

As noted by Wieczorek, Hoheisel and Prodan [260], more recently, a relatively small 
number of workflow management and execution frameworks were proposed, which 
admitted a more expressive application model than the DAG and targeted the 
computational grid. The grid in this sense is a distributed system of resources, that can be 
applied to reach a common goal, e.g., enable faster processing of a complex task. Taxo-
nomically, it is a uniform or, more commonly, non-uniform heterogeneous scheduling 
platform. Extended application models are used within the ASKALON framework 
[73], which is based on the Abstract Grid Workflow Language (AGWL) [74], and in 
K-WfGrid, based on Petri nets [91]. However, for the purpose of scheduling the work-
flow is always reduced to a DAG in both approaches.

With the advent of cloud computing platforms the trend toward employing only well-
studied and simple application models was intensified. In contrast to scheduling ap-
proaches proposed in the fields of classical parallel processing and in the grid, very 
often those which target the cloud argued against application-centric optimization ob-
jectives, such as the schedule length or the quality of service (QoS) of a workflow in-
stance, which can also be viewed as end user-centric criteria. The focus largely shifted 
to resource-centric optimization objectives, such as resource utilization or energy effi-
ciency, which are, by contrast, service provider-side efficiency criteria. From this point 
of view, expressive application models are clearly less important. Cloud schedulers di-
rectly inspired by previous approaches targeting the grid and application-centric QoS 
continued to rely on the DAG model [21]. However, a substantial body of work in 
contemporary cloud scheduling treats the bag-of-tasks (BoT), i.e., a set of independent 
jobs, as the application model of choice [43, 177].

All the results reported refer to sets of tasks with deterministic processing times, linked 
by constraints that, as shown in Sect. 5.2.1, may be represented by appropriately cho-
sen, restricted subclasses of the Petri net model. For the unrestricted model, van der 
Aalst, van Hee and Reijers [241] provided results that are arguably of most interest 
here. They focused on discrete-time stochastic Petri nets, in which two classes of tran-
sitions exist. The first includes those that may fire immediately (as in ordinary Petri
nets). The members of the second subset are associated each with a single discrete stochastic variable, interpreted as execution time of the event represented by the transition. The following elementary configurations (subnets) were taken into account: a sequence, an ordinary parallel execution (AND-parallelism), an exclusive choice (XOR-parallelism) and a loop, where both of the latter have fixed probabilities known a priori. Formulas were provided to compute the discrete distributions of the execution time of these subnets, based on the assumption that the stochastic variables involved are independent and thus a convolution algorithm can be applied. To reduce its running time, the use of discrete Fourier transforms (DFTs) was postulated. For the loops, only the expected values and coarse bounds on the standard deviation (non-directional dispersion) were derived, the latter based on Chebyshev’s inequality.

These results inspired a substantial body of work aimed at the derivation of the overall distribution of quality-of-service values in similar nets. As it is known [260], in general, different QoS parameters that may be simultaneously associated with a set of task graph nodes have each its own aggregation model with respect to the elementary subgraphs. For example, quality and reliability are usually assumed to be multiplicative in sequences and minimal across parallel branches [273]. One of the most comprehensive list of formulas describing the aggregation of QoS values in Petri net-reducible task graphs was given by Hwang et al. [112], who filled in the gaps left by van der Aalst, van Hee and Reijers [241] for most of the other aggregation models encountered in practice.

All of the approaches just reported can be employed to compute an approximate probability distribution of the runtime, or a set of attributes, of a project described by a given combination of elementary subgraphs. When only runtime is considered, this is the extension of the PERT makespan distribution computation problem (see 2.15.1) to task graphs admitting conditional (only exclusive-OR) branches and loops. Notably, the solutions are still approximate, as in general the assumption of independence does not hold for the stochastic variables involved. Moreover, the PERT-like problem so formulated is still at least #P-hard. This makes it impossible to directly apply this approach in any algorithm that requires repeated estimation steps for partial execution paths. Thus, in particular, this approach neither solves a proper scheduling problem, which additionally entails allocation, nor provides means to solve it efficiently.
5.5 Solution Approach

For the $G/G/c : Q|v_i, c_{ij} \sim \text{stoch, wn}|E[W]$ problem, a scheduling policy is proposed whose general structure is outlined in Alg. 14. It shares some similarities with the DAG scheduling approach proposed in the previous chapter (TOFF/ϵ-M, Alg. 10). In fact, both are online, non-preemptive, elementary scheduling policies (Line 4). Moreover, both implement a modified list scheduling paradigm, where specific deviations from the order given by the priority list are enforced by a parameter-controlled greediness mechanism. This mechanism is designed similarly to that of the Alg. 10. Controlled greediness means a priority-driven only (lazy) behavior at all times with the exception of prioritizing a lower priority job over a higher priority one which will only be available in the future, when certain conditions are met. A local look-ahead is performed to determine how scheduling the job of lower priority immediately may improve its current mapping and to what extent it may delay the job of higher priority. These two estimates are used to compute a trade-off value which additionally takes into account the priorities of the jobs with respect to the original priority list. A threshold value (Line 1) controls how large trade-off values are accepted (Line 12) and is an input parameter of the policy (as in Alg. 10 different parametrizations may be used). In general, the larger this value is, the more greedy the policy is, hence the designation “parameter-controlled greediness”. This policy will be referred to as TOFF/WN-M, i.e., Trade-off-based Workflow Net scheduler for Multiple instances.

Apart from those similarities, however, the application model is different. A consequence of the adoption of the workflow net model is the fact that the determination of the set of schedulable jobs, which was trivial for DAG-reducible applications, becomes inherently difficult. With workflow nets, the execution monitor’s knowledge of the presence of a workflow instance in the system and of the set of jobs scheduled so far is not sufficient to determine the set of potentially schedulable jobs. Firstly, given the conditional control flow, some jobs within the instance may be skipped altogether. Intuitively, they should not be considered for scheduling nor in the priority list determination, when it becomes apparent that the branches containing them have been discarded. Considering them would mean unnecessary computation and may lead to inaccurate prioritizing and wrong decisions with respect to the greedy-versus-lazy problem. On the other hand, in the presence of loops in the application model, a job might have to run again, even if it has already been scheduled. Thus, jobs belonging...
5.5. Solution Approach

Algorithm 14: A generic formulation of the proposed TOFF/WN-M policy.

1. **input:** an instance of $G/G/c : Q|v_i, e_{ij} \sim \text{stoch}, \text{wn}|E[W]$; a threshold value $\epsilon$; optionally a maximum queue length $N$

2. **output:** a feasible schedule

3. **while** maximum queue length not exceeded do

4. wait until a job of an instance $I$ becomes available

5. compute the set of schedulable jobs $J \triangleright J$ contains only one realization for all jobs not flagged as inactive

6. compute the job priority list $L = L(J, I) \triangleright$ only update the list for $I$

7. initialize the table of potential mappings $M$

8. **while** $L$ has unexamined jobs do

9. consider the next highest priority job $j$ in $L$

10. **foreach** processor $p$ in the pool do

11. compute the estimated earliest finish time mapping $m(j, p)$

12. cache $m(j, p)$ in $M$

13. initialize the best mapping $m^*(j, p)$ for $j$

14. **foreach** mapping $(m(j, p)) \in M$ in decreasing order of finish time do

15. **foreach** mapping of each job $j'$ of higher priority than $j$ do

16. estimate the delay caused to the finish time of $j'$ by scheduling $j$ according to $m(j, p)$ and vice-versa

17. compute a trade-off between the delays

18. **if** trade-off below the threshold **then**

19. $m^*(j, p) \leftarrow m(j, p)$

20. remove all other mappings for $j$ from $M$ and break both **foreach** loops

21. **if** $j$ can already be executed according to $m^*(j, p)$ **then**

22. schedule $j$ according to $m^*(j, p)$

23. flag as inactive all jobs alternative to $j$ and their children, up to the next OR-join having a job with no flag

24. remove $m^*(j, p)$ from $M$

A generic way of determining the set of schedulable jobs is a full reachability analysis of the marked workflow net, given the current marking. However, whereas updating the marking is a trivial task for a runtime monitor, the reachability problem in workflow nets is $\text{PSPACE}$-hard at best, i.e., assuming that the net is guaranteed to be 1-safe by design. Moreover, for any marking for which there exist reachable loops containing a transition corresponding to a job $j$, the number of different realizations of job $j$ is potentially infinite (in Petri net terms, the reachability graph is infinite). This makes intractable any approach that would require priority computation across the set of all job realizations corresponding to reachable transitions.
These observations lead to a balanced algorithm design based on the following principles. After the completion of a job, when the next conditional branches to be executed are determined (in Petri net terms, when a place with multiple output transitions is reached), the jobs corresponding to the transitions in the discarded conditional branches are flagged as inactive up to the next OR-join in the net (Alg. 14, Line 16). If all jobs connected to the join are flagged inactive, the procedure continues beyond the join. This procedure will be termed simple propagated elimination: “simple” because it eliminates from the schedulable job set all the jobs that are definitely unreachable, without an extensive search in the space of all future system states or net markings. An place with multiple input transitions is considered to be marked (have a token) if it has been reached through all transitions corresponding to the jobs that are not flagged inactive. The loops that cannot be reached in any future state of the workflow instance are entirely eliminated as well. For any remaining potentially reachable loop, only a single future job realization for all the transitions belonging to the loop is considered at a given scheduling step, unless flagged as inactive due to the elimination procedure outlined above (Alg. 14 Line 5). In addition to this potential future realization of job \( j \), the information about all of the actual (past and present) realizations of \( j \) is naturally preserved.

Obviously, the application of critical path- or level-based prioritizing heuristics to jobs in workflows that do not conform to the DAG model is also difficult. The following sections illustrate the related issues and provide details of the mathematical framework and algorithmic solution approaches. These approaches are valid under the assumption that the underlying model is the workflow net, which is structured and can be partitioned into sequences, loops, parallel and alternative branches (see Sect. 5.2.1). This assumption constitutes a limitation of all methods proposed, as they cannot be applied directly to arbitrary Petri net models which are not reducible to a workflow net.

### 5.5.1 Length Estimation in Stochastic Conditional Paths

As outlined in Sect. 5.2.1, the exclusive choice (XOR) pattern can be modelled, in a 1-safe Petri net, by a set of transitions competing for a single token in a common preset place. This is illustrated in Figure 5.4 where, given a token in place \( p \) any single
transition $t_i$ may fire (a single job $j_i$ associated with that transition may be executed). It is assumed (Sect. 5.5) that each job $j_i$ has a known distribution of runtime, associated with a stochastic variable $X_i$. Additionally, the condition which must be true for $t_i$ (job $j_i$) to be chosen may have a known prior probability $a_i$. When the condition is evaluated in $p$, a single transition (job) is chosen and the length of the path leading from $p_s$ to $p_e$ becomes known. However, before this happens, the length of this path (set of paths) can be estimated. Estimating it is also necessary to accurately prioritize the jobs within the workflow net: for example, the set of paths in question may or may not be a part of the critical path in the net. The following theorem provides the means for this estimation.

![Figure 5.4: A Petri net with a set of conditional branches.](image)

**Theorem 4.** Let there be a set $\{X_i\}$ of stochastic variables, where each $X_i$ has a probability density function (PDF) $f_i(x)$. Let there be a set of independent events $E_i = \{X = X_i\}$, where each $E_i$ has a known probability $a_i$ and $X$ is another stochastic variable. Let $\bigcup_i E_i$ be the whole event space and let there be an event $E = \{X \in B\}$, where $B \subseteq \mathbb{R}$ is a value range of $X$.

Then the PDF of $X$ is given by the sum of the PDFs of the variables $X_i$, weighted by the probabilities of the events $E_i$:

$$f(x)dx = \sum_i a_i f_i(x)dx$$  \hspace{1cm} (5.2)

**Proof.** By Def. 2.19 $Pr[E] = Pr[X \in B] = \int_B f(x)dx$. By the assumptions above, the set of events $\{E_i\}$ is a finite partition of the sample (event) space. Therefore, by the law of total probability, it holds that $Pr[X \in B] = \sum_i Pr[\{X \in$
Because the events \( \{X_i \in B\} \) and \( \{X = X_i\} \) are independent, it is also true that \( Pr[\{X_i \in B\} \cap \{X = X_i\}] = Pr[X_i \in B] Pr[X = X_i] = a_i \int_B f_i(x) dx \) and also, clearly, \( \sum_i Pr[\{X_i \in B\} \cap \{X = X_i\}] = \sum_i a_i \int_B f_i(x) dx \). Thus it holds that \( \int_B f(x) dx = Pr[E] = \sum_i a_i \int_B f_i(x) dx \). Because the integration operator is linear, it also must hold that \( f(x) = \sum_i a_i f_i(x) \).

It is easy to see how the result above is applicable in the workflow net context. Let there be two transitions and two jobs corresponding to them. If the probability that \( j_1 \) requires 10 seconds to execute is 0.5 and the probability that \( j_2 \) requires the same amount of time is 0.25, what is the probability that, if either \( j_1 \) or \( j_2 \) is executed, this process will take 10 seconds to complete? Clearly this event is as likely as choosing \( j_1 \) and obtaining a 10 second-long execution, or choosing \( j_2 \) and obtaining the same result. Thus, for each support point of the probability density function (discrete or continuous), the probabilities add up, scaled by the probabilities of the respective conditions (exclusive choices).

If the variable \( X \) is defined as in the theorem above, the partial expectation of \( X \) over an interval \([y, z]\) is given by:

\[
E_y^z[X] = \int_y^z x f(x) dx = \sum_i a_i \int_y^z x f_i(x) dx = \sum_i a_i E_y^z[X_i] \tag{5.3}
\]

and the marginal expectation of \( X \) is its partial expectation over the interval \((-\infty, +\infty)\):

\[
E[X] = \mu = \sum_i a_i E[X_i] \tag{5.4}
\]

The \( n \)-th partial moment of \( X \) about a point \( x_0 \), over the interval \([y, z]\), is given by:

\[
E_y^z[(X - x_0)^n] = \sum_{k=0}^{n} \binom{n}{k} (-x_0)^k E_y^z[X^{n-k}] = \sum_{k=0}^{n} \binom{n}{k} (-x_0)^k E_y^z[\sum_i a_i X_i]^{n-k} \tag{5.5}
\]

In particular, the positive semi-variance of \( X \), i.e., the second moment of \( X \) about the marginal mean \( \mu \) over the interval \([\mu, +\infty)\), is given by:

\[
Var_+[X] = \sigma_+^2 = \int_{\mu}^{\infty} (x - \mu)^2 f(x) dx = \sum_i a_i \int_{\mu}^{\infty} (x - \mu)^2 f_i(x) dx \tag{5.6}
\]
The negative semi-variance of $X$ is defined analogously and it holds (see Eq. 3.8):

$$Var[X] = \sigma^2 = Var_+[X] + Var_-[X] = \sigma_+^2 + \sigma_-^2$$

(5.7)

where $\sigma^2$ is the (total, non-directional) standard deviation and $\sigma_+^2$, $\sigma_-^2$ are the positive and negative semi-deviations, respectively.

To model the non-exclusive OR pattern, we assume an execution semantics for the net in Figure 5.4 (see Sect. 5.2.1). The above example reasoning may be adapted to the new pattern. What is the probability that, if either $j_1$ or $j_2$ is executed, or both are, this process will take 10 seconds? This event is as likely as choosing $j_1$ and not $j_2$, and obtaining a 10 second-long execution, or choosing $j_2$ and not $j_1$, or choosing both, and obtaining the same result. The only difference with the above scenario is the fact that there are now three independent events, for which the probabilities are not expressed directly by the input values $a_1, a_2$, but need to be derived, as these events were previously implicit. This is straightforward if one considers that, under the assumption of independence of two events $E_j, E_k$, it holds that $Pr[E_j \land E_k] = Pr[E_j]Pr[E_k] = a_1 a_2$, $Pr[E_j \land \neg E_k] = Pr[E_j](1 - Pr[E_k]) = a_1(1 - a_2)$. Importantly, to estimate the length of the path set in question, it is not necessary to model the set of independent events $\{E_i\}$ in the net, i.e., no transformation of the net is needed. In fact, for large values of $n$ and given that the transitions corresponding to the jobs $j_1, j_2$ could be replaced with complex subnets, such a transformation would result in a very large output net. Instead, the length of each branch can be computed (or estimated) as for any set of OR- or AND-parallel branches and then the reasoning above can be applied to the obtained stochastic variables or their estimates.

Given explicit independent events, the execution length of OR-joint branches (branches that correspond to non-exclusive choices) may be formulated employing the same approach and formulas already derived for the XOR pattern. Without this reformulation, the computation of the length of the path leading to an OR-join would require the determination of the maximum of a set of non-independent random variables, which would make the approach more complex.

The previous discussion refers to offline distributions and estimates, i.e., is valid under the assumption that the execution of the subnet containing the two conditional branches of interest has not commenced yet (or that no information about its state of execution
is available). If, on the other hand, it is known that one of the activities in one of the branches has been commenced, it is also clear which branches have been chosen for execution in the current workflow instance: the others have been discarded. The online distribution or the relative estimate is thus obviously much easier to compute. For running (incomplete) activities within the active branch the conditional partial mean estimation method known from Chapter 4 Sect. 4.4.1 can be applied, while the length of those still to be executed can be estimated using the marginal partial moments method (Chapter 3 Sect. 3.5.1).

5.5.2 Length Estimation in Iterative Stochastic Paths

Let us consider a loop, modelled, in a Petri net, by a pair of transitions \( t_1, t_2 \), as in the subnet in Figure 5.5 that correspond to the execution of two jobs \( j_1, j_2 \). For simplicity, let us assume that the durations of \( t_1, t_2 \) are two independent stochastic variables \( X, Y \), with \( f_x(x), f_y(x) \) the respective probability density functions (PDFs). For clarity we also assume, without loss of generality, that there are no communication delays associated with the precedence constraints (which are apparent from the figure).

![Figure 5.5: A simple Petri net with a loop.](image)

Clearly, \( t_1 \) will be executed at least once with probability equal to one, after which the execution of the entire sub-net may terminate, with probability \( 1 - a \), or the sequence \( t_1, t_2 \) will be executed, with probability \( a \), where \( a \in \mathbb{R} \) with \( 0 < a < 1 \) (strictly not equal as otherwise this would not be a proper loop). Let \( Z(n) \) be the stochastic variable that describes the total duration of the loop, when executed \( n \) times. Then
5.5. Solution Approach

\[ Z(n) \text{ is given by:} \]

\[ Z(n) = X + a(Y + X) + a^2(Y + X) + \ldots + a^n(Y + X) = X + \sum_{i=1}^{n} a^i(Y + X) \quad (5.8) \]

**Theorem 5.** Given \( a \in \mathbb{R} \) with \( 0 < a < 1 \) and the stochastic variable \( Z(n) \) as defined in Eq. (5.8), the following holds:

\[ \lim_{n \to \infty} Z(n) = X + \frac{1}{1-a}(X + Y) = \frac{2-a}{1-a}X + \frac{1}{1-a}Y = Z_\infty \quad (5.9) \]

**Proof.** Given \( a < 1 \), each term \( a^{i+1}(Y + X) \) is less significant than \( a^i(Y + X) \). For \( n \to \infty \) the sum \( \sum_{i=1}^{n} a^i \) is convergent and equal to \( \frac{1}{1-a} \). The stochastic variables \( X \) and \( Y \) are independent of the number of iterations \( i \), whether it is finite or not.

The following can be observed, on the basis of Eq. (5.9). For a theoretically infinite number of iterations, the total length of the loop described by the stochastic variable \( Z_\infty \) is equivalently given by the sum of two independent random variables \( X_1, X_2 \), linked to the original variables \( X, Y \) of the problem by the following equations:

\[ X_1 = \frac{2-a}{1-a}X, \quad X_2 = \frac{1}{1-a}Y \quad (5.10) \]

\( X_1, X_2 \) are termed supplementary variables, as they do not appear in the original problem. Notably, for the two original variables associated with the transitions in the loop, only two supplementary variables are required. This reasoning extends in a straightforward manner to the case in which there are communication delays associated with the edges connecting transitions \( t_1, t_2 \), or there are more transitions. The supplementary variable is derived by choosing the appropriate coefficient (here: \( \frac{2-a}{1-a} \) or \( \frac{1}{1-a} \)) based on whether the associated activity precedes or follows the exit point of the loop (place \( p_c \) in Figure 5.5).

Given the properties of the expectations of random variables, both the total and the partial marginal expected values of \( Z_\infty \) can be easily derived. Higher moments and conditional moments of any order would require additional computation steps to determine the CDF (or, equivalently, the PDF) of \( Z_\infty \) first. In Chapter 2, Eq. (2.29) an exact solution for the this problem was given. Throughout this thesis computationally efficient methods of deriving the needed estimates have already been employed.

189
(Sect. 3.5.1 Sect. 4.4.1), also taking into consideration the fact that, during a realistic scheduling process, there are very few cases in which the random variables to be added are truly independent (see 2.15.1).

Let us consider an activity which immediately follows an exit point of a loop, such as $t_e$ in Figure 5.5. The length of the path leading to it across the loop depends on the loop length given by $Z_\infty$ (plus the length of the path leading to the loop entry). From the point of view of this activity, it does not matter whether $Z_\infty$ is formulated as an infinite sum or otherwise: The activity outside the loop only perceives the effective loop length, not its structure. Thus for the purpose of estimating start and finish times of an activity outside the loop, the original loop structure can be replaced by a sequence of two artificially inserted activities with durations given by $X_1, X_2$. This virtual sequence can then be treated like any other sequence of activities and, in this context, $Z_\infty$ can be treated in the same manner as other sums of random variables. It is only necessary to replace the original variables $X, Y$ of the problem with $X_1, X_2$. The reformulation of the problem by means of supplementary variables which are still mutually independent makes it possible to apply estimation methods with a configurable complexity. This contrasts with the approaches reported in Sect. 5.4 ([241] [112]), in which computing the probability density of the runtime for a net is at least $\#P$-hard. Moreover, the reduction of loops to sequences enables a consistent employment of the familiar techniques for the determination of task levels and critical paths in a graph. In particular, non-trivial loops, such as partially overlapping loops, can be easily supported in this manner.

Let us consider Figure 5.6: there are two possible entry points (places) $p_{s1}, p_{s2}$ and two possible exit points $p_{e1}, p_{e2}$, each of the latter associated with a probability of remaining in the loop $a_1, a_2$ or exiting from it. From the point of view of the entry place $p_{s1}$, the formula for the variable $Z_n$ describing the length of $n$ iterations of the loop becomes:

$$Z_n = X_1 + a_1 X_2 + a_1 a_2 X_3 + a_1 a_2 X_1 + a_1^2 a_2 X_2 + a_1^2 a_2^2 X_3 + ..$$

(5.11)

Therefore, if $e$ is the counting variable over all the $e_{max}$ exit points of the loop, in general it holds:

$$Z_n = \sum_{i=1}^{n} \left( \prod_{e=1}^{e_{max}} a_e \right)^i \sum_{e=1}^{e_{max}} \frac{X_e}{\prod_{t=e}^{e_{max}} a_t}$$

(5.12)
The reasoning can be repeated for any other entry point. Overlapping loops can model checks that are performed multiple times and can be interrupted at several points: As such, they are actually common in realistic workflows. Importantly, in the proposed approach, they are as easily tractable as other loops. Instead of the variables \( X_1, X_2 \) derived for the simple loop, we now have a set of \( e_{max} \) supplementary random variables given by:

\[
\frac{X_e}{\prod_{t=e}^{e_{max}} a_t}
\]  

(5.13)

There are as many supplementary variables as there were variables in the original problem. Let:

\[
\prod_{e=1}^{e_{max}} a_e = A
\]

(5.14)

As in the simple loop case, the term \( \sum_{i=1}^{n} A^i \) is independent of these variables and, for \( n \to \infty \), converges to \( \frac{1}{1-A} \). It is easy to see that it will be close to one for most values of \( a_i \).

Figure 5.6: A Petri net subnet with overlapping loops.

Van der Aalst, van Hee and Reijers [241] assigned a constant probability to the event corresponding to exiting from the loop (or staying in it) in each iteration, and therefore a memoryless property of the loop: In Eq. 5.8 above these are the values \( 1 - a, a, \)
respectively. Hwang et al. \cite{112}, on the other hand, assumed that the number of iterations be described by a discrete distribution with a finite support. In our approach, neither assumption is necessary: Unlike in Hwang et al. \cite{112}, the number of iterations may be theoretically infinite even with a non-constant \( a = a(i) \), where \( i \) is the iteration number. It is not even necessary that \( a(i) \) be non-increasing. The values assumed by the stochastic variable \( Z_\infty \) describing the total length of the loop will still be finite, therefore the representation of the loop by means of a virtual sequence will still be meaningful. In fact, as \( a(k) < 1 \) for any \( k \), the sum \( \sum_{i} a_{i}^{\text{max}} \) converges even for \( a_{\text{max}} = \max_k a(k) \), i.e., for the largest of all \( a(k) \) values. Therefore, and more so, it must converge if in some of the \( a_{i}^{\text{max}} \) terms \( a_{\text{max}} \) is replaced with a smaller value. If, on the other hand, the loop has a bounded number of iterations, the term \( \sum_{i=1}^{n} a_{i} \), which describes a memoryless loop (with a constant \( a \)), becomes \( \sum_{k=1}^{n} a_{i} \), where \( n \) is the last support point for \( a(k) \), i.e., \( \forall n' > n \Rightarrow a(n') = 0 \).

The above discussion introduced means of approximating the offline distribution of the length of a loop. In an online scheduling setting, the execution monitor is able to determine two additional situations, based on the marking of the workflow net. In the first situation, a loop may become inactive for the current workflow instance. Informally, this happens when the control flow “moves on” beyond a state in which the loop can be activated. In other words, given the current marking, there is no reachable marking in which any transition belonging to the loop is enabled. In the simple propagated elimination approach, it corresponds to a situation in which all jobs in the loop have been flagged as inactive by executing Line 16 of Alg. \cite{14}. In this case the actual loop length is known.

The second situation is one in which there is an active transition in the loop, that has been executing for a known amount of time \( \tau \). In this case, it is possible, but also somewhat challenging, to improve the estimates of the length of the active loop. On the one hand, the remaining time in the current loop execution has a stochastic length that depends on \( \tau \), but also, clearly, on which activity is being executed. On the other hand, if the loop is memoryless, the number of potential future executions of the loop is still unknown, as it was before the first execution of the loop. Therefore, the potential future executions of the loop contribute to the stochastically distributed total loop length according to the same principle and in the same manner as specified by Eq. \ref{eq:5.8} (in practice, for this purpose the counting variable \( i \) is zeroed). The reason is the
5.5. Solution Approach

fact that in this case the probabilities \((a, 1 - a)\) do not depend on the number \(i\) which describes how many times the loop has been executed up to now.

Let us assume first that \(t_1\) has been executing for the amount of time given by \(\tau\). Let \(X'\) be the stochastic variable describing the remaining execution time of \(t_1\), i.e., \(X' = (X|X \geq \tau) - \tau\).

Let \(Z(X', n)\) be the stochastic variable that describes the remaining total duration of the loop, when executed \(n\) times, given that it is known that the remaining execution time of \(t_1\) is described by \(X'\). Then \(Z'(X', n)\) is given by:

\[
Z'(X', n) = X' + a(Y + X) + a^2(Y + X) + \ldots + a^n(Y + X) = X' + \sum_{i=1}^{n} a^i(Y + X) \tag{5.15}
\]

and furthermore:

\[
Z'(X', n) = Z(n) + X' - X \tag{5.16}
\]

Therefore:

\[
Z'_{\infty}(X') = \lim_{n \to \infty} Z'(X', n) = X' + \frac{1}{1 - a}(X + Y) = Z_{\infty} + X' - X \tag{5.17}
\]

Therefore, the total remaining length of the loop in this case is equivalently described by the sum of three independent random variables \(X', X_3, X_4\), where:

\[
X_3 = \frac{1}{1 - a} X, \quad X_4 = \frac{1}{1 - a} Y = X_2 \tag{5.18}
\]

Let us assume now that \(t_2\) has been executing for the amount of time given by \(\tau\). Let \(Y'\) be the stochastic variable describing the remaining execution time of \(t_2\), i.e., \(Y' = (Y|Y \geq \tau) - \tau\).

Let \(Z(Y', n)\) be the stochastic variable that describes the remaining total duration of the loop, when executed \(n\) times, given that it is known that the remaining execution time of \(t_2\) is described by \(Y'\). Then \(Z'(Y', n)\) is given by:

\[
Z'(Y', n) = Y' + X + a(Y + X) + a^2(Y + X) + \ldots + a^n(Y + X) = Z(n) + Y' \tag{5.19}
\]
Therefore:

\[ Z'_\infty(Y') = \lim_{n \to \infty} Z'(Y', n) = Y' + \frac{a}{a - 1} X + \frac{1}{1 - a} Y = Z_\infty + Y' \quad (5.20) \]

Therefore, the total remaining length of the loop in this case is equivalently described by the sum of three independent random variables \( Y', X_1, X_2 \), as defined above.

The distribution of the remaining length of an active loop can always be computed using a procedure analogous to the two cases discussed above and the relative equations. If the loop in question is not memoryless, in an online framework it is possible to update all of the above estimates knowing \( a(i) \) and \( i \).

### 5.5.3 Priority Attributes in Workflow Nets

Within the framework of workflow nets and given the computation methods outlined in the previous sections, it is possible to reformulate the classical prioritizing methods in list scheduling within the stochastic workflow net context. Algorithm 15 shows an adaptation of the top level computation method to this context.

One such modification relates to the case in which a parent task is connected to a loop exit point and a child is outside of that loop (Lines 14-17). From the point of view of the child, the length of the sequence leading to it corresponds to the length of the loop, incremented by the largest top level of any of the loop’s entry tasks. In the previous section a method of estimating this length has been proposed, based on a transformation of the loop into a virtual sequence of activities. Other methods are possible: for example, it may be conjectured that, most of the time, the loop will execute only up to the exit point (optimistic assumption) or that it will complete at least one full cycle (see Sect. 5.6), hence the presence of the policy modifier \( \Pi_l \) (Lines 1 and 16).

A second modification relates to the presence of a set of conditional paths connecting a parent and children tasks. During the execution of an instance, each time a choice is made (an OR-split resolved, Line 8), any such set collapses to a known subset of paths and the online prioritizing algorithm will take advantage of this knowledge. However, before this happens, the length of the path set leading to an OR-join must be estimated (Lines 10, 18-19): Sect. 5.5.1 has outlined an estimation method based on appropriate...
weights, assigned to each path. Other methods are possible (hence the provision for a policy modifier $\Pi_c$, Line 10): in particular, one that assumes the maximum of all the estimated lengths of the relevant paths, i.e., the worst-case alternative.

The remaining part of the algorithm (Lines 3-8, 11-13, 21-24) corresponds to the well known t-level computation procedure for directed acyclic graphs. It is trivial as it follows directly from Def. 2.47 and Eq. 5.1.

**Algorithm 15:** Top level priority ordering with conditionals and loops.

1. **input:** a list of schedulable jobs $J$ in topological order; a list of structural loops $\Lambda$; a strategy of treating conditional branches $\Pi_c$ and loops $\Pi_l$
2. **output:** a job priority list
3. **while** $J$ has unexamined jobs **do**
   4. select the next job $j$ and the associated transition $t$
   5. **foreach** place $p \in \bullet t$ **do**
      6. $J_p \leftarrow \{ \}$ **▷** initialize the active parent job set
      7. **foreach** parent transition $t_p \in \bullet p$ **do**
         8. add to $J_p$ each job $j_p$ associated with $t_p$, if not flagged inactive
      9. **if** $|J_p| = |\bullet p|$ **then**
         10. **optionally**, assign to each $j_p$ a weight determined according to $\Pi_c$
      11. $T(J_p) \leftarrow 0$ **▷** initialize the top level variable for $J_p$
   12. **foreach** job $j_p \in J_p$ **do**
      13. look up the top level $t_l(j_p)$
      14. **foreach** loop in $\Lambda$ containing $t_p$ and not $t$ **do**
         15. look up the top level of the loop entry node
         16. estimate the loop length according to $\Pi_l$
         17. assign the sum of the above values to $t_l(j_p)$
      18. **if** $j_p$ has an assigned weight **then**
      19. multiply $t_l(j_p)$ by the weight and add the result to $T(J_p)$
      20. **else**
         21. replace $T(J_p)$ with $t_l(j_p)$, if the latter is larger
      22. estimate the delay $d(j_p, j)$, add it to $T(J_p)$ and assign this value to $t_l(p)$
      23. assign to $j$ the maximum value of $t_l(p)$ as its top level
   24. order all jobs according to increasing top levels

The bottom level prioritizing algorithm is analogous. For the dominant sequence priority, once the b-levels are known, the critical path can be built choosing for each node, starting with the entry node, its child with the largest b-level. The addition of the other
nodes to the sequence can be then performed highest b-level first, as in the dominant sequence building method for directed acyclic graphs (see Sect. 2.16.1).

Alg. 15 employs the information about the workflow net to estimate path lengths for the purpose of assigning priority attributes to jobs. Note that the algorithm requires the knowledge of the list of the loops present in the structure of the workflow net. As the structure is static, the determination of this list may be performed only once for each net, prior to the scheduling phase. This problem can be solved trivially by applying a slight modification of Tarjan’s strongly connected components algorithm for directed graphs [228]. Assuming that all the places and transitions of the net are mapped to vertices in a directed graph, we obtain an algorithm of worst-case complexity linear in the sum of the number of places, transitions and arcs of the original net.

5.5.4 Complexity Analysis

This section evaluates the complexity of the scheduling policy proposed for the $Q|v_i, c_{ij} \sim \text{stoch}, wn | E[W]$ problem. Given that an online policy for potentially infinite queues (and scheduling decisions) is considered, only the complexity of a single algorithm call is investigated. The exact complexity ($\Theta$) is provided for the steps for which it is available, the worst-case ($\hat{O}$) complexity for the remaining cases.

As indicated in Sect. 3.5.3, the complexity of the computation of each partial moment of the probability distribution of a random variable representing an activity is linear in the number $s$ of the support points in the discrete representation of the CDF. In addition to the variables corresponding to the $a$ activities (job execution times and communication delays) originally present in the application’s model, the supplementary variables introduced for all the loops need to be considered. However, the supplementary variables are in fact the original variables of the problem with previously derived, constant coefficients. Thus they do not influence the complexity of this step of the algorithm, which is indicated below.

$$\hat{\Theta} = 2sa$$

(5.21)

If the conditional moment-based estimation method is employed, the observation just made for the determination of partial moments holds and total complexity of the prior-
ity list computation becomes (see 4.4.3):

$$\hat{\Theta}_{\text{cond}} = 2s(a + 2r)$$  \hspace{1cm} (5.22)

In the equation above, \(r\) is the total number of activities that are currently executing (\(r_v\) being the number of jobs and \(r_e\) that of inter-task communications). It depends on the previous behavior of the specific scheduling algorithm employed and on the problem instance. In the presence of loops, it is not bounded by the total number of activities in the application model. However, a bound can still be formulated, based on the total number of processors and communication channels:

$$r = r_v + r_e, \quad r_v \leq p, \quad r_e \leq p^2$$  \hspace{1cm} (5.23)

If \(u\) is the number of jobs yet to be scheduled, the worst-case complexity of the trade-off-based mapping evaluation is given by:

$$\hat{O} = pu(u - 1), \quad u \leq nv - r_v$$  \hspace{1cm} (5.24)

where \(v\) is the total number of jobs in the application model, \(r_v\) the number of jobs currently running and \(n\) is the number of active instances. The latter can be approximated, for a stationary queue, according to Little’s law (Eq. 4.13), as

$$n \approx E[n] = \lambda E[W].$$

As in the cases investigated in the previous chapters, the total complexity is a function of the priority list determination algorithm employed (which is unrelated to the estimation methods and the trade-off-based refinements). In the given online multi-instance context, there may be many jobs of different instances to prioritize. However, given the assumption of a natural FIFO instance priority, the jobs belonging to different instances are naturally partitioned, with a number of partitions equal to the number \(n\) of the active instances in the system. The only remaining complexity problems are those related to the employed job prioritizing methods within an instance.

In the classical DAG application model, the b- and t-level job prioritizing methods are worst-case quadratic, while the dominant sequence is worst-case cubic in the number of nodes in the task graph (see Sect. 3.5.3). From Alg. 15 it is apparent that, in a workflow net, the computation of the top level of a job also depends on the number of
active alternative parent jobs and on the number of loops containing any parent job. If \( l \) is the number of structural loops in the net, the overall worst-case time complexity of the top level priority ordering is given by the following equation.

\[
\hat{O}_t = v^3(1 + l)
\]

Clearly, the effective complexity is much lower, as, in practice, the number of AND- and OR-splits relative to the set of the parents of a job is much lower than the total number \( v \) of task nodes in the application. Similarly, the number of loops relevant for a particular job is a fraction \( l \). When applied in an online policy, in practice, the method also acts on sets of parents and loops with a decreasing number of members. This is due to the elimination of the parents that are flagged as inactive due to the choices made during the execution of the instance. The b-level computation has exactly the same (worst-case and effective) complexity, while the dominant sequence is more complex by a factor of (at most) \( v \).

The total worst-case complexity of the TOFF/WN-M policy with conditional partial moment-based estimators and a dominant sequence priority list is given by:

\[
\hat{O}_{total} = nv^4(1 + l) + 2s(a + 2r) + pu(u - 1) \leq nv^4(1 + l) + 2s(v + e + 2r^2) + pn^2v^2
\]

5.6 Experimental Evaluation

For evaluation purposes, the procedure outlined in Sect. 2.19 was employed. Two sets of 100 workflow nets each were created and, for each single net from the above set, 100 realizations, for a total of 20000 different workflow instances. The first set encompasses nets with conditional branches but no closed loops, while in the second structural loops are admitted as well. In the latter case, the number of loops is limited: There is one such loop, or two, with equal probability.

Each of the nets has an arbitrary structure, randomly generated by adapting the procedure used previously (Sections 3.6, 4.5). The relevant parameter values were chosen
5.6. Experimental Evaluation

as follows:

1. number of transitions (corresponding to jobs) \( v \in \mathbb{N} \), chosen randomly from the interval \([10, 20]\),

2. maximum alpha \( \alpha_{max} \) of 0.25, 1, 4; the actual values of \( \alpha \) uniformly distributed in the interval \((0, \alpha_{max}]\), where \( \alpha \sqrt{v} \) is the mean graph height,

3. maximum postset size of a transition (corresponding to the out-degree in a DAG) of 5 and the actual size uniformly distributed in the interval \([1, \text{max}]\),

4. maximum postset size of a place (corresponding to the number of alternative branches) of 5 and the actual size uniformly distributed in the interval \([1, \text{max}]\),

5. a marginal probability associated with each combination of edges connecting a place \( p \) with multiple transitions uniformly distributed in \((0, 1)\); if \( p \) is an exit point of a loop, then the split is prescribed to be exclusive (XOR),

6. maximum level jump (number of places between connected transitions) of 5 and actual jump uniformly distributed in \([1, \text{max}]\); this parameter can also assume negative values, in which case it determines the maximum length of a loop,

7. mean job execution times \( w(\cdot) \) sampled uniformly randomly from \([1, 10]\),

8. computation-to-communication ratio (CCR) of 0.25, 1, 4.

The transitions generated in 1 are partitioned into levels, each with a number of transitions determined by \( \alpha \) (2). When a place \( p \) in the postset of transition \( t \) is connected to this transition \( t \), the level jump parameter (6) is used to determine the target level, which contains a child transition \( t_c \). From the set of transitions which share that level, \( t_c \) is chosen at random. Place \( p \) is then connected to \( t_c \) as a preset place. This first part of the procedure mirrors the generation of random directed acyclic graphs, with places and arcs added appropriately to obtain a correct DAG-equivalent Petri net.

Conditional control flow, on the other hand, requires multiple transitions to share an input place. Therefore, the second phase of the generation procedure processes the places in the net that are permitted to have child transitions (all but the sink place). At this stage, each such place \( p \) already has exactly one transition in its postset. The

199
cardinality of the postset may be changed to be larger than one (4). In this case, the first child transition \( t_c \) is already known. The remaining ones are determined according to (6). All the arcs originating in \( p \) are then assigned marginal probabilities (5). An additional mechanism is employed to guarantee that the output be a correct workflow net. Let \( t_c \bullet \) be the postset of \( t_c \). Each new child transition \( t'_c \) beyond \( t_c \) is connected directly to all places in \( t_c \bullet \) and the arcs linking \( t'_c \) to its former postset places are removed. The postset places are removed as well, if “orphaned” (not connected to any other transition) at this point. This routine removes some of the complexity in the net’s structure, but it ensures that an XOR-split in \( p \) has a corresponding XOR-join. Furthermore, we assume that any place \( p' \in t_c \bullet \) must be reachable for all the reachable markings originating with a single token in \( p \). In practice, we require that all the AND- and XOR-splits in the subnet between \( p \) and \( p' \) must have their corresponding joins in that subnet. The splits for which this is not true are removed with their associated arcs and places. Note that for the soundness of the output workflow net these requirements are, in general, not necessary but they are sufficient. This set of steps is meant to be a reasonably simple mechanism to construct a workflow net that is guaranteed to be correct. Analogously, for the construction of subnets with loops, it is required that all splits have their corresponding join within the subnet contained between each entry and exit place of the loop.

As in the previous chapters, the mean job execution times \( w(\cdot) \) are taken from a normal distribution with the given support and used to generate completely arbitrary continuous four-parameter Beta distributions (see Appendix A). Each such distribution has a mean \( \mu = w(\cdot) \) and is supported in \([\mu, 4\mu]\) and has a normalized standard deviation \( \sigma = \mu \) of a value from the set \{0.01, 0.25, 0.5, 0.75, 1, 1.25\}. The communication delays are assigned four-parameter Beta distributions extrapolated analogously and in a manner such that the requirement given by the CCR factor is satisfied.

For each graph and number of processors, an estimate \( \lambda_{\text{max}} \) of the critical arrival rate is computed in a pre-simulation phase. A lazy scheduling policy with a baseline FIFO and dominant sequence priority (see Table 5.1) is used to schedule a batch of 10 instances of each graph onto the available scheduling platform. The average makespan determines the sample-based expectation on the service rate under this policy. This is an approximation of the maximum value of the arrival rate that would not result in an increasing number of instances waiting for service (an unstable queue), if this
5.6. Experimental Evaluation

particular lazy policy were employed to schedule the entire queue. Note that, for the whole set of the policies evaluated here, this estimated baseline rate is conservative. In fact, we expect any non-lazy counterpart of the reference policy to perform better in terms of the expected makespan objective by leaving fewer idle time slots and thus allow for larger service rates. The actual values of the arrival rate $\lambda$ are calculated as $\lambda = \lambda_{mul}\lambda_{max}$, where the factor $\lambda_{mul}$ is chosen from $\{0.25, 0.5, 0.75, 1, 1.25, 1.5\}$. In the light of the above explanation, it is possible for some policies to schedule a potentially infinite queue under an arrival rate given by the critical arrival rate of the baseline lazy policy, multiplied by a factor larger than one (given that the estimate of the critical arrival rate is pessimistic). If, however, the arrival rate is excessive, i.e., the queue becomes unstable, the results should indicate this. We expect a comparatively large difference of the sample-based expectation on the response time between the rates for which the queue is stable and those for which it is not.

The number of processors $|Q|$ is also defined in a parametrized manner. The actual number of processors is determined multiplying the number $v$ of transitions (jobs) in the workflow net by a set of factors $p_{mul} \in \{0.25, 0.5, 0.75, 1\}$ as follows: $|Q| = \lceil p_{mul}v \rceil$. The processor speeds are chosen as uniform random values in the interval $(0, 1]$, as are the speeds of the processor-to-processor physical channels. The speeds of the channels are not correlated with the speed of the processors they connect. No physical channel has a “zero” speed (the platform is fully connected).

The performance evaluation is based on the methodology outlined in Sect. 2.19 and has already been applied in Chapter 4. In particular, the main performance metric adopted is the sample mean of the normalized response time, given in the equation below.

$$E[W_n](\pi) = \frac{1}{N} \sum_{i=0}^{N} \frac{W(i, \pi)}{\sum_{j \in J_{CP}(i)} r(i, w(j))}$$

(5.27)

For each scheduling policy $\pi$, the values of the response time $W(i, \pi)$ are determined, for $N$ workflow instances. The sampled response time for each instance $i$ is normalized by the overall time needed for the processing of the instance’s critical path (the sum $\sum_{j \in J_{CP}(i)} r(i, w(j))$ in Eq. 5.27, where $r(i, w(j))$ is the realization of job $j$ in instance $i$). The performance metric for each policy is the mean of these normalized response times. We can observe that a completed instance of a workflow net is a set
of realizations of tasks and inter-task communication edges, linked by ordinary precedence constraints. At runtime, all conditions are ultimately verified and each OR-split is resolved and becomes the origin of a single actual path or multiple AND-parallel paths. Each loop is also unfolded into a sequence, with as many realizations of each task as required (until an exit path from the loop is chosen). In other words, a completed instance is simply a DAG with deterministic activity times (a different one for each instance) and, therefore, has a well defined critical path, which makes the sum \( \sum_{j \in J_{CR}(i)} r(i, w(j)) \) well defined as well.

In addition to the mean \( E[W_n(\pi)] \), shown on the graphs as a circle, for the normalized response time \( W_n(\pi) \), the statistical dispersion of its values is also shown, given by the appropriate five-number summaries (104). Additionally, wherever a variant of the trade-off-based scheduling policy appears to differ only slightly in terms of performance with respect to a competitor, a Mann-Whitney \( U \) test (164, also known as Wilcoxon rank-sum test) is performed to establish the significance of the performance difference, which is shown in separate tables. The robustness metric is the aggregation of the positive and negative partial deviations of the response times, derived and normalized similarly. The performance is shown with respect to the following independent parameters (horizontal axes): the processor multiplier (\( p_{mul} \)), the arrival rate multiplier (\( \lambda_{mul} \)) and the normalized standard deviation (\( nsd \)) of the stochastic variables determining the duration of the activities in the application.

We investigate the performance impact of different strategies of estimating the length of conditional branches and loops, when applied to a representative range of heuristics, referenced throughout this thesis. Two larger experiment groups are thus summarized in the following. In the first, workflow nets with OR-splits and joins are scheduled, adopting stochastic estimates derived based on one of two strategies. The first strategy (\( w_{XOR} \)) assigns a weight to each of the paths in an OR-join according to the prior probabilities of all possible independent paths with the method derived in Sect. 5.5.1. As this approach becomes complex for multiple non-exclusive OR-splits, a much simpler alternative is tested: \( m_{XOR} \), which discards all paths but the longest one, independently of its prior probability. In other words, the \( m_{XOR} \) strategy effectively treats the OR-split like an AND-split of all branches.

The second group of experiments comprises the application of three possible strategies to the problem of estimating the length of paths associated with closed loops. The
length of a reachable loop contributes to determining the priority of each job associated with a transition connected to an exit point of the loop. Sect. 5.5.2 has outlined a rigorous method, based on the equivalence of the potentially infinite loop to a virtual sequence of activities: It bears the suffix \texttt{vseq} when applied to any other combination of heuristics. In this strategy, the priority of a transition (job) connected to a loop exit point is derived with the method outlined in Sect. 5.5.2. Specifically, a set of supplementary random variables is determined according to Eq. 5.13. Two more straightforward approaches are examined as well. The \texttt{sum} strategy assumes the loop length to be simply equivalent to the sum of the durations of all activities within the loop, independently of any entry and exit. In other words, \texttt{sum} effectively performs a simple loop unravelling procedure (Sect. 5.2) once. The \texttt{min} strategy, on the other hand, follows the optimistic assumption that the loop will be executed only once and only up to the exit point associated with the transition of interest. On average, we expect \texttt{sum} to overestimate and \texttt{min} to underestimate the loop length, while the experiments should show if there are cases in which adopting either of these more efficient methods instead of \texttt{vseq} is justified.

The estimation strategies described above are applied to two sets of scheduling policies: thus, both experiment groups are also further divided into two sets (Table 5.1). The first set of each group shows the performance of scheduling policies characterized by different degrees of greediness. It includes the corner cases (the best performers among the lazy and greedy algorithms, indicated as \texttt{LZY} and \texttt{GDY}) and the idle time-charged policy family (\texttt{ITC-E=k}). The latter represents a parameter-driven approach to greediness: The value of the parameter \(k\) determines the maximum time a job of lower priority must wait in the presence of jobs of higher priority that are not available for scheduling. The results for ITC are shown separately for readability. The second set of each group comprises state-of-the-art heuristics for heterogeneous environments, with their specific machine assignment and speed normalization methods: \texttt{HEFT} [233], \texttt{SHEFT} [227] and \texttt{DLS} [220]. To determine job priorities within each workflow instance, for DLS the dynamic level attribute is prescribed, whereas both HEFT and SHEFT are assumed to employ downward ranks (bottom levels computed with the default average speed normalization strategy, see Sect. 2.16.1). The results show SHEFT with its stochastic estimator of choice (sum of mean and standard deviation, \texttt{MSD}), HEFT with the marginal mean (\texttt{MM}) and DLS with the conditional partial moment estimator (\texttt{CM}), with which it performed slightly better than with ei-
ther MSD or MM. In both sets, TOFF/\(WN\)-M, i.e., the trade-off-based policy outlined in Alg. 14 is included: The second set features only TOFF with the best estimation strategy determined based on the performance in set 1. TOFF employs CM estimation, dominant sequence priority and no speed normalization, as do all policies in set 1. The trade-off value accepted by TOFF decreases with the growing arrival rate of new instances into the system (see Table 5.1). Any value is accepted when the finish time of instances of higher priority is unlikely to be delayed, i.e., the out-branch node delaying mechanism is employed as in the case of Alg. 8.

<table>
<thead>
<tr>
<th>Sets</th>
<th>Features</th>
<th>Algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>1, 3</td>
<td>Greediness</td>
<td>LZY/([S]): lazy algorithm (Alg. 2), with a strict FIFO and dominant sequence (D) priority; no speed normalization; conditional partial moment (CM) estimation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>GDY/([S]): greedy algorithm (Alg. 1), priority-driven: baseline FIFO/D priority but no deliberate idle time; no speed normalization; CM estimation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ITC-E=(k)/([S]): idle time-charged algorithm (Alg. 5) with the cost-charging parameter set to (k), baseline FIFO/D priority; no speed normalization; CM estimation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TOFF/([S]): multi-instance trade-off-based algorithm TOFF/(WN)-M with a variable threshold of (0.1/\lambda_{mul}), active out-branch node delay; baseline FIFO/D priority; no speed normalization; CM estimation</td>
</tr>
<tr>
<td>2, 4</td>
<td>Priority and machine assignment heuristics</td>
<td>HEFT/([S]): baseline FIFO queue, downward rank with average speed normalization; marginal mean (MM) estimation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>SHEFT/([S]): baseline FIFO, downward rank with average speed, SHEFT estimation (sum of marginal mean and standard deviation, MSD)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>DLS/([S]): baseline FIFO, dynamic level based on b-level with average speed, CM estimation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TOFF: TOFF algorithm with the best performing XOR/loop handling strategy from set 1</td>
</tr>
</tbody>
</table>

Table 5.1: Outline of the experiments for \(G/G/c : P/Q|v_i, c_{ij} \sim \text{stoch, wn}|E[W]\).
5.6. Experimental Evaluation

<table>
<thead>
<tr>
<th>Sets</th>
<th>Features</th>
<th>Strategies ([S])</th>
</tr>
</thead>
</table>
| 1,2  | OR-split/join | wXOR: prior probability-based weighting of potentially active conditional paths  
|      |           | mXOR: maximum length across conditional paths |
| 3,4  | Loop     | vseq: virtual sequence weighting of potentially active loops  
|      |           | sum: sum of activity durations in one unravelled loop for potentially active loops  
|      |           | min: minimal path to exit for potentially active loops |

Table 5.2: Outline of the experiments for \( G/G/c : P/Q|v_i, c_{ij} \sim stoch, wn|E[W] \), continued.

All the policies examined follow a baseline FIFO queue discipline of workflow instances, whereby LZY is the only policy to follow this order strictly. The others may deviate from it to various degrees, interleaving jobs belonging to different instances, as determined by the algorithm’s logic: For example, GDY schedules the first available job of any instance moving down through the priority list. To make the comparison fair, all policies recompute job priorities taking advantage of information available at runtime: in particular, when unreachable branches and loops are discarded.

5.6.1 Discussion

The results in the initial set of the first experiment group (Figure 5.7-5.10) show a substantial difference in performance in favor of the TOFF scheduling policy, with either branch handling strategy (wXOR or mXOR), when compared to the policies that exhibit a purely lazy (LZY) or mostly lazy behavior (ITC-E=0.5). This difference is less significant, although still in favor of TOFF, for greedier algorithms (ITC-E=0.1, ITC-E=0.05) and the outright greedy policy (GDY): These perform consistently better than their lazier counterparts throughout most of the examined parameter value ranges.

In general, the wXOR strategy benefits the TOFF scheduling policy substantially, except in the high range of the processor multiplier \( p_{mul} > 0.6 \), and slightly less the greediest among the other polices: GDY and ITC-E=0.05. The remaining, lazier schedulers benefit more from mXOR, especially at low ranges of the processor multiplier, arrival rate multiplier and normalized standard deviation of activity times. In par-
Chapter 5. Scheduling of Petri Net-reducible Stochastic Workflows

In particular, \texttt{mXOR} yields a better performance at $p_{\text{mul}} < 0.5$, $\lambda_{\text{mul}} < 1.5$ and $nsd < 1.25$ and is more robust for $nsd < 1.25$ than \texttt{wXOR} for all lazier policies (\texttt{LZY}, \texttt{ITC-E=0.5}). This is the exact opposite of the behavior of \texttt{GDY}, for which the \texttt{wXOR} strategy is preferable in the same parameter value ranges.

![Graph](image)

Figure 5.7: Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function: (a) of the processor multiplier $p_{\text{mul}}$, given $|P| = \lceil v p_{\text{mul}} \rceil$; (b) of the multiplier $\lambda_{\text{mul}}$ of the critical arrival rate, given $\lambda = \lambda_{\text{mul}} \lambda_{\text{max}}$. Algorithms with variable greediness.

206
Figure 5.8: (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation \( nsd = \frac{\sigma}{\mu} \) of the activity durations; (d) standard deviations of the normalized response time, as a function of \( nsd \). Algorithms with variable greediness.
Table 5.3: Statistical significance of the performance difference between TOFF/wXOR and TOFF/mXOR. Two-tailed Mann-Whitney U test: sample size $10^4$; * = significance at $p < 0.05$, ** = significance at $p < 0.01$.

<table>
<thead>
<tr>
<th>$p_{mul}$</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1</th>
<th>1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>$p$-value</td>
<td>0.0083</td>
<td>0.0083</td>
<td>0.0407</td>
<td>0.0196</td>
<td>0.0021</td>
<td>0.092</td>
<td>0.0832</td>
<td>0.0830</td>
<td>0.0866</td>
<td>0.0734</td>
</tr>
<tr>
<td>significance</td>
<td>**</td>
<td>**</td>
<td>*</td>
<td>*</td>
<td></td>
<td>*</td>
<td>*</td>
<td>**</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$\lambda_{mul}$</td>
<td>0.25</td>
<td>0.5</td>
<td>0.75</td>
<td>1</td>
<td>1.25</td>
<td>1.5</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p$-value</td>
<td>0.0103</td>
<td>0.0093</td>
<td>0.0103</td>
<td>0.0071</td>
<td>0.0085</td>
<td>0.5958</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>significance</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td>**</td>
<td></td>
<td>**</td>
<td></td>
<td>**</td>
<td></td>
<td></td>
</tr>
<tr>
<td>$nsd$</td>
<td>0.01</td>
<td>0.25</td>
<td>0.5</td>
<td>0.75</td>
<td>1</td>
<td>1.25</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>$p$-value</td>
<td>0.9822</td>
<td>0.0192</td>
<td>0.0192</td>
<td>0.0587</td>
<td>0.0104</td>
<td>0.0078</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>significance</td>
<td>*</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td>**</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the second set of the first group additional effects may be observed, resulting from the combination of specific machine assignment mechanisms, mixed greedy-lazy behavior patterns and the methods employed to estimate stochastic activity times. The online DLS policy appears to be almost insensitive to the choice between the wXOR and mXOR strategy, with the latter slightly preferable for very low processor multiplier and very high normalized standard deviation ranges ($p_{mul} = 0.2$, $nsd = 1.25$). Shown on the graphs is the best-performing version of DLS, equipped with the conditional partial-moment based stochastic estimator, originally devised for TOFF. This variant of DLS, while in general inferior to TOFF, provides a level of performance almost equal for low values of $nsd$ ($nsd < 0.75$) and in the high processor multiplier range ($p_{mul} > 0.6$), where it slightly outperforms TOFF in a single instance ($p_{mul} = 1$).

In both variants of the heterogeneous earliest finish time heuristic, HEFT and SHEFT, the mXOR strategy performs always better than wXOR, except in the middle range of the processor multiplier ($p_{mul} \in \{0.7, 0.8\}$), at the opposite ends of the arrival rate range ($\lambda_{mul} \in \{0.25, 1.25\}$) and in the low $nsd$ range ($nsd < 0.75$), where the strategies are almost indifferent. As expected, both variants of HEFT perform similarly for low normalized standard deviations of the input stochastic variables, when the MSD estimator (applied in SHEFT) yields values close to the marginal mean, i.e., derived with the MM estimation method. Both HEFT policies, independently of their stochastic estimates, exhibit an overall performance visibly inferior to TOFF and DLS, except in the low $nsd$ range ($nsd < 0.5$). In one instance (only for $nsd$ values close to zero) HEFT/mXOR and SHEFT/mXOR are slightly more robust than their competitors. At $nsd > 0.75$, the HEFT policy performs poorly both in terms of the response time.
5.6. Experimental Evaluation

mean value and robustness.

Figure 5.9: Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function: (a) of the processor multiplier $p_{mul}$, given $|P| = \lceil vp_{mul} \rceil$; (b) of the multiplier $\lambda_{mul}$ of the critical arrival rate, given $\lambda = \lambda_{mul} \lambda_{max}$. Algorithms with variable greediness.
Figure 5.10: (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $\text{nsd} = \frac{\sigma}{\mu}$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $\text{nsd}$. Algorithms with parameter-driven greediness.
5.6. Experimental Evaluation

Figure 5.11: Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function: (a) of the processor multiplier $p_{mul}$, given $|P| = \lceil vp_{mul} \rceil$; (b) of the multiplier $\lambda_{mul}$ of the critical arrival rate, given $\lambda = \lambda_{mul} \lambda_{max}$. Algorithms with different job prioritizing and machine assignment heuristics.
Figure 5.12: (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $nsd = \sigma / \mu$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $nsd$. Algorithms with different job prioritizing and machine assignment heuristics.
5.6. Experimental Evaluation

Summarizing, wXOR is mostly of advantage for greedier scheduling policies, but also for algorithms such as TOFF that are particularly sensitive, by design, to accurate estimations of path lengths. Unlike in greedy algorithms, in TOFF wXOR is never worse than mXOR: It may be indifferent in settings with a large number of processing units. Given that applying mXOR reduces complexity, in such cases this strategy may be an alternative. It is also the best strategy, in most cases, in lazy policies and both the dynamic level-based algorithm and the examined variants of the heterogeneous earliest finish time heuristic.

The second group of experimental results indicates that, in the TOFF policy, the strategy consisting in treating loops of activities as virtual sequences (vseq) is almost always preferable to the alternatives considered. There are two exceptions to this rule at the opposite ends of the nsd range considered: the (optimistic) strategy assuming the minimal path to the loop’s exit (min) performs visibly better for nsd ≈ 0 and, for nsd = 1.25, it is substantially more robust, although slightly inferior to vseq in terms of the mean of the normalized response time. In large processor pools (p_mul > 0.7) the strategies min and sum are indifferent in TOFF, while vseq has a similar performance but is slightly more robust. In smaller pools and in most of the range of the arrival rate multiplier and normalized standard deviation of activity durations, with the exceptions mentioned above, sum performs slightly better than min. In GDY, vseq and sum are almost indistinguishable, except at the higher end of the nsd range, where vseq performs slightly better and is more robust. In LZY, vseq has the best performance, except at lower arrival rates (λ_mul ≤ 0.75), where sum delivers the smallest response times. The min strategy appears to be completely inadequate for the lazy scheduling

<table>
<thead>
<tr>
<th>p_mul</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1</th>
<th>1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-value</td>
<td>0.0364</td>
<td>0.0071</td>
<td>0.0080</td>
<td>0.0484</td>
<td>0.0087</td>
<td>0.0093</td>
<td>0.0107</td>
<td>0.0093</td>
<td>0.0093</td>
<td>0.0078</td>
</tr>
<tr>
<td>significance</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>λ_mul</th>
<th>0.25</th>
<th>0.5</th>
<th>0.75</th>
<th>1</th>
<th>1.25</th>
<th>1.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-value</td>
<td>0.0082</td>
<td>0.0084</td>
<td>0.0068</td>
<td>0.0102</td>
<td>0.0078</td>
<td>0.0116</td>
</tr>
<tr>
<td>significance</td>
<td>**</td>
<td>**</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td>**</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>nsd</th>
<th>0.01</th>
<th>0.25</th>
<th>0.5</th>
<th>0.75</th>
<th>1</th>
<th>1.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-value</td>
<td>0.0341</td>
<td>0.0073</td>
<td>0.0370</td>
<td>0.0233</td>
<td>0.0093</td>
<td>0.0082</td>
</tr>
<tr>
<td>significance</td>
<td>*</td>
<td>**</td>
<td>*</td>
<td>*</td>
<td>**</td>
<td>**</td>
</tr>
</tbody>
</table>

Table 5.4: Statistical significance of the performance difference between TOFF/wXOR and DLS/mXOR. Two-tailed Mann-Whitney U test: sample size $10^4$; * = significance at $p < 0.05$, ** = significance at $p < 0.01$.}

213
policy, a fact that is confirmed by a similar behavior of the “next laziest” algorithm examined, **ITC-E=0.5**.

![Figure 5.13](image)

**Figure 5.13:** Sample mean and dispersion of the response time for an instance, normalized by the processing time of all the task realizations in the instance’s critical path, as a function: (a) of the processor multiplier $p_{mul}$, given $|P| = \lceil v_{p_{mul}} \rceil$; (b) of the multiplier $\lambda_{mul}$ of the critical arrival rate, given $\lambda = \lambda_{mul}\lambda_{max}$. Algorithms with different greediness.
Figure 5.14: (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $nsd = \frac{\sigma}{\mu}$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $nsd$. Algorithms with different greediness.
In the latter algorithm, \texttt{vseq}, once again, derives solutions dominating those obtained by means of the competing strategies in all cases. In lazier members of the idle time-charged family (\texttt{ITC-E=0.05, ITC-E=0.1}), such dominance is given for lower arrival rates ($\lambda_{mul} < 1.25$), while the differences are more significant in \texttt{ITC-E=0.05} and for lower normalized standard deviations. For arrival rates higher than one, the \texttt{min} strategy performs better, by a margin depending on the greediness of the algorithm (the factor $k$ in \texttt{ITC-E=k}). This explains why the overall behavior of ITC algorithms as a function of the processor multiplier does not show either of these strategies to be consistently better.

The following result set, split in two parts for readability, compares the best-performing policy in the first set, i.e., \texttt{TOFF/vseq}, with all combinations of policies derived through applying the possible loop path length determination strategies to the DLS and HEFT reference schedulers. The dynamic level scheduling algorithm (\texttt{DLS}) clearly benefits from the \texttt{vseq} strategy as well. At $nsd = 1.25$, the strategies are indifferent. In one instance (at $p_{mul} = 0.4$), the \texttt{min} strategy is slightly better than \texttt{sum}, while throughout the rest of the parameter value ranges examined it is dominated by \texttt{sum}.

Overall, as in the first experiment group which investigated OR-join-handling strategies, the best member of the DLS family of policies, i.e., \texttt{DLS/vseq}, performs better than any variant of the heterogeneous earliest finish time heuristic. However, \texttt{TOFF/vseq} still outperforms \texttt{DLS/vseq} in all cases, except at very low arrival rates ($\lambda_{mul} = 0.25$), where DLS is slightly better in terms of the response time mean, and at very high values of $nsd$, where there is no significant difference. The margin in favor of TOFF decreases with a growing processor multiplier.

\begin{table}[h]
\centering
\begin{tabular}{|c|c|c|c|c|c|c|c|c|}
\hline
$p_{mul}$ & 0.2 & 0.3 & 0.4 & 0.5 & 0.6 & 0.7 & 0.8 & 0.9 & 1 & 1.1 \\
\hline
$p$-value & 0.0080 & 0.0162 & 0.0062 & 0.0068 & 0.0093 & 0.0102 & 0.0102 & 0.0093 & 0.0083 & 0.0089 \\
\hline
significance & ** & * & ** & ** & * & * & * & ** & ** \\
\hline
\hline
$\lambda_{mul}$ & 0.25 & 0.5 & 0.75 & 1 & 1.25 & 1.5 \\
\hline
$p$-value & 0.0081 & 0.0080 & 0.0090 & 0.0078 & 0.0089 & 0.0145 \\
\hline
significance & ** & ** & ** & ** & * \\
\hline
\hline
$nsd$ & 0.03 & 0.25 & 0.5 & 0.75 & 1 & 1.25 \\
\hline
$p$-value & 0.0079 & 0.0091 & 0.0082 & 0.0068 & 0.0079 & 0.0063 \\
\hline
significance & ** & ** & ** & ** & ** & ** \\
\hline
\end{tabular}
\caption{Statistical significance of the performance difference between TOFF/vseq and TOFF/sum. Two-tailed Mann-Whitney $U$ test: sample size $10^4$; * = significance at $p < 0.05$, ** = significance at $p < 0.01$.}
\end{table}
5.6. Experimental Evaluation

Figure 5.15: Sample mean and dispersion of the response time for an instance, normalized by the processing time of all the task realizations in the instance’s critical path, as a function: (a) of the processor multiplier $p_{mul}$, given $|P| = \left\lceil vp_{mul} \right\rceil$; (b) of the multiplier $\lambda_{mul}$ of the critical arrival rate, given $\lambda = \lambda_{mul}\lambda_{max}$. Algorithms with parameter-driven greediness.
Figure 5.16: (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation \( \text{nsd} = \frac{\sigma}{\mu} \) of the activity durations; (d) standard deviations of the normalized response time, as a function of \( \text{nsd} \). Algorithms with parameter-driven greediness.

In HEFT, \text{vseq} is always a markedly better strategy and adopting it results in very large gains with respect to the others especially for larger arrival rate and \( \text{nsd} \) values. The sum strategy dominates min, with a few exceptions: \( \lambda_{mul} = 1, \text{nsd} > 0.75 \). In SHEFT, the differences attributable to the loop estimation strategies are much smaller and there are more cases in which \text{vseq} is outperformed (\( p_{mul} \in \{0.2, 0.4\}, \lambda_{mul} = 1.5, \))
nsd = 1.25), but it remains the best strategy on average. The best performers in the HEFT and SHEFT series deliver similar results, with HEFT leading in the lowest processor multiplier range ($p_{mul} < 0.4$), SHEFT leading in the sub-range $0.3 < p_{mul} < 0.6$ and an indistinguishable performance elsewhere. SHEFT is generally more robust than either HEFT or DLS, especially in the higher nsd range ($nsd > 0.5$).

Summarizing, the vseq strategy is mostly preferable to any other and sum usually dominates the simple optimistic strategy min. When optimality may be sacrificed for efficiency, a less complex DLS policy might be an acceptable alternative to the best performing one, i.e., TOFF/vseq, but only in the presence of a sufficiently large number of processing units.

Overall, it may be noted that the differences between estimation strategies for both OR and structural loop constructs are less significant in certain experimental settings. This is especially true in three cases. If the number of processing units in the system is relatively large, accurate estimation becomes less important. If the uncertainty associated with the input values is large, accurate estimation of the path lengths becomes impossible because of the dispersion of the values of the stochastic variables, independently of the way they are added. If the relative arrival rate of new workflow instances into the system is excessive, a lazier policy cannot deliver an acceptable performance level, independently of the stochastic estimation and path aggregation methods employed.

<table>
<thead>
<tr>
<th>$p_{mul}$</th>
<th>0.2</th>
<th>0.3</th>
<th>0.4</th>
<th>0.5</th>
<th>0.6</th>
<th>0.7</th>
<th>0.8</th>
<th>0.9</th>
<th>1</th>
<th>1.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-value</td>
<td>0.0080</td>
<td>0.0219</td>
<td>0.9011</td>
<td>0.0095</td>
<td>0.2866</td>
<td>0.0074</td>
<td>0.0084</td>
<td>0.0130</td>
<td>0.0089</td>
<td>0.0079</td>
</tr>
<tr>
<td>significance</td>
<td>**</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$\lambda_{mul}$</th>
<th>0.25</th>
<th>0.5</th>
<th>0.75</th>
<th>1</th>
<th>1.25</th>
<th>1.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-value</td>
<td>0.0198</td>
<td>0.0093</td>
<td>0.0095</td>
<td>0.0182</td>
<td>0.0548</td>
<td>0.0103</td>
</tr>
<tr>
<td>significance</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td>*</td>
<td>*</td>
<td>**</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>nsd</th>
<th>0.01</th>
<th>0.25</th>
<th>0.5</th>
<th>0.75</th>
<th>1</th>
<th>1.25</th>
</tr>
</thead>
<tbody>
<tr>
<td>p-value</td>
<td>0.0084</td>
<td>0.0087</td>
<td>0.0126</td>
<td>0.0076</td>
<td>0.0065</td>
<td>0.633</td>
</tr>
<tr>
<td>significance</td>
<td>**</td>
<td>**</td>
<td>*</td>
<td>**</td>
<td>**</td>
<td>**</td>
</tr>
</tbody>
</table>

Table 5.6: Statistical significance of the performance difference between TOFF/vseq and DLS/vseq. Two-tailed Mann-Whitney $U$ test: sample size $10^4$; * = significance at $p < 0.05$, ** = significance at $p < 0.01$.  

219
Figure 5.17: Sample mean and dispersion of the response time for an instance, normalized by the processing time of all the task realizations in the instance’s critical path, as a function: (a) of the processor multiplier $p_{mul}$, given $|P| = \lceil v_{p_{mul}} \rceil$; (b) of the multiplier $\lambda_{mul}$ of the critical arrival rate, given $\lambda = \lambda_{mul}\lambda_{max}$. Algorithms with different job prioritizing and machine assignment heuristics.
5.6. Experimental Evaluation

Figure 5.18: (c) Sample mean and dispersion of the response time for an instance, normalized by the processing time of the task realizations in the instance’s CP, as a function of the normalized standard deviation $nsd = \frac{\sigma}{\mu}$ of the activity durations; (d) standard deviations of the normalized response time, as a function of $nsd$. Algorithms with different job prioritizing and machine assignment heuristics.

All policies are capable of recomputing the path lengths relevant for the purpose of determining job priorities and estimating their finish times. Thus, from a certain moment during the execution of a workflow instance onwards, the length of each path associated with an OR or loop construct becomes exactly known. This has the effect of reducing the number of possible outcomes of, e.g., a priority list determination rou-
tine, because there are fewer values that the algorithm must “guess” according to some estimation strategy. In essence, within the same policy group (core algorithm, e.g., DLS), the behavior resulting from adopting different estimation strategies becomes more similar with the progress of the execution of a workflow instance.

Figure 5.19: Sample mean and dispersion of the response time for an instance, normalized by the processing time of all the task realizations in the instance’s critical path, as a function: (a) of the processor multiplier $p_{\text{mul}}$, given $|P| = \lceil vp_{\text{mul}} \rceil$; (b) of the multiplier $\lambda_{\text{mul}}$ of the critical arrival rate, given $\lambda = \lambda_{\text{mul}}\lambda_{\text{max}}$. Algorithms with different job prioritizing and machine assignment heuristics.
5.7 Summary

In this chapter, we proposed a scheduling policy for the problem of minimizing the expected response time when a potentially infinite queue of workflow instances is sched-
Chapter 5. Scheduling of Petri Net-reducible Stochastic Workflows

uled onto a uniformly heterogeneous platform. Both the activity durations within the workflow and the instance arrival process are assumed to be given by arbitrary stochastic variables, with the respective distributions only subject to very mild assumptions. In particular, it is required that the mean of the arrival rate be known and that the activity durations (task execution and inter-task communication times) have finite first and second partial moments in the sense of Sect. 4.4.1.

At the same time, we observed that many realistic workflows require a broader range of control flow constructs than are expressible in the directed acyclic graph (DAG) model, adopted in the previous chapters and in most scheduling approaches for sets of inter-dependent jobs. In particular, an application model is needed that explicitly supports conditional control flow (choice) and iterative flow (loops), in addition to the series-parallel flow possible in DAGs. The workflow net model is chosen as fundamentally similar and close to the minimal superset of the DAG model that allows for the inclusion of choice and iteration. Workflow nets are structured workflows and a subclass of ordinary Petri nets, a formally well-founded and well-studied model of concurrency and control flow, which is amenable to analysis and for which a rich tool set is available.

The introduction of conditional and iterative flow into the application model adds to the complexity of the scheduling problem defined above. The scheduling policy must make decisions without knowing which execution paths will be chosen for a particular workflow instance and how many times a particular loop will be re-entered. In other words, the scheduler has no exact knowledge of the set of jobs that are to be executed or of the path lengths that determine the priorities of the jobs.

For the sub-problem of determining the set of executable jobs, we proposed a heuristic, called simple propagated elimination. Upon a choice of a conditional branch, it labels as inactive the first jobs in all the discarded branches and all the jobs within those branches which only have inactive-labelled parents: This procedure typically stops at the next OR-join that corresponds to another unresolved OR-split (a choice not made yet). For the sub-problem of determining priorities within the set of potentially active jobs, we proposed an extension of the list scheduling approach. We followed the assumption that more accurate estimation strategies of the length of a set of alternative paths or of a loop connected to a workflow net transition representing a job results in a more accurate determination of the priority and of the start and finish times of the job.
5.7. Summary

While these heuristics do not provide guarantees, we expect proven methods such as dominant sequence prioritizing and trade-off-based mapping to deliver the best results, on average, when equipped with more accurate path estimation strategies.

Given these premises, for the problem of estimating the length of multiple alternative paths (before any branch is chosen), a method has been proposed which is based on weighting the paths of interest in accordance to the prior probabilities assigned to each branch. For non-exclusive choices, these probabilities must be additionally derived by artificially introducing the corresponding exclusive choices, i.e., considering any combination of choices, but the workflow net does not need to be modified for this purpose. For loops, we put forward a method that is based on transforming the loop into a virtual sequence of activities. The new activities are described by a set of supplementary random variables, linked to the original ones by coefficients derived based on the probabilities of exiting the loop, or remaining in it, and they take into account possible multiple exit points of a loop.

Both methods make it possible to approximate entire distributions of the random variables describing the path lengths, given the random variables associated with the activities within each path. Clearly, for reasons of efficiency, in the proposed scheduling policy (and the alternatives examined) the distributions of the variables of interest are replaced with scalar estimates, e.g., their partial moments, and the path aggregation strategies are applied to these estimates only.

The proposed policy schedules then the available jobs according to their priorities. When a job of higher priority is not available and it is unlikely to be delayed by a decision to schedule greedily a job of lower priority, the latter is scheduled, if it can thus finish earlier.

This approach was thoroughly evaluated using a procedure that generates a large number of application models, where each is a workflow net with a random structure. The method assigns then completely arbitrary continuous four-parameter Beta distributions to the activities in the application. Two sets of models were generated: with and without iterative flows (closed loops). The proposed policy was demonstrated to have the best performance and robustness in most cases, when compared with state-of-the-art schedulers for heterogeneous platforms, and several other variations of list algorithms of similar complexity. As these algorithms were originally designed for the DAG application model, they were provided with the same mechanisms that enable scheduling
Chapter 5. Scheduling of Petri Net-reducible Stochastic Workflows

of workflow nets in the policy proposed in this chapter. More efficient path length estimation strategies were subjected to experimental evaluation as well, which allowed for identifying the settings in which such less complex methods may be alternatively applied, at a reasonably small expense in performance. In particular, in settings with a relatively small uncertainty of the inputs (a small normalized standard deviation of the activity runtimes) and with large sizes of the processor pool, simpler heuristics can be applied, such as dynamic level scheduling. In the same settings simple strategies are effective in estimating the length of a set of conditional paths, e.g., the longest estimate-based path. Conversely, when relatively scarce processing resources are available or the uncertainty of the inputs is substantial, the proposed policy is always preferable. The estimator based on the idea of weighting conditional branches according to their probabilities also brings a significant advantage to the trade-off-based policy in these problem settings. The adoption of the strategy which treats loops of activities as virtual sequences is especially advantageous when the relative size of the processor pool is small to medium with respect to the size of the set of jobs in the application. This strategy also benefits the most the trade-off-based policy, but in many cases it also delivers the best results when applied to other scheduling heuristics.
Chapter 6

Conclusions

This thesis investigates a range of problems which arise when workflows with stochastic activities are scheduled onto a platform comprised of homogeneous or uniformly heterogeneous processors and communication channels connecting them.

It focuses on time-domain objectives that are the primary concern of any customer on whose behalf the workflow is executed. The response time of the system is one such concern. If the system is dedicated to that particular customer, the response time coincides with the execution time of the workflow instance. If the system is shared with other users, the response is given by the sum of the execution time of the instance belonging to the user and its in-queue waiting time.

Many simpler workflows are defined as a definite number of partially ordered sets of activities and thus their control flow can be formulated by means of the classical directed acyclic model. This is considered as the underlying control flow part of the application model for both a single-instance problem setting and one setting that features multiple workflow instances. However, a distinctive feature of all the application models and problem settings considered in this thesis is the fact that both task execution and inter-task communication times are assumed to be described by random variables with arbitrary distributions, which must satisfy only mild requirements.

For the single-instance setting, an offline scheduling policy is formulated which, under the assumptions outlined above, is more generally applicable than any competing approach. The policy minimizes the expected schedule length over a large number of workflow instances. For the multi-instance setting, an online policy is proposed
to schedule a queue of workflows, containing likewise structured stochastic activities. The queue originates from a stochastic arrival process, which may be completely arbitrary, provided that its average rate is known. The policy minimizes the expected response time of the system. It is more effective than competing online algorithms which only consider jobs that are ready for execution and employ simple rules to select a subset of them. It is also more broadly applicable than competing approaches utilizing look-ahead strategies, as it does not make restrictive assumptions regarding the probability distributions describing the activity times or the instance arrival process.

Many realistic workflows also require that conditions be evaluated at runtime and choices be made with respect to the set of activities to execute and their order of execution, i.e., feature conditional execution paths. Given a set of paths, the choice may be exclusive (only one such path) or non-exclusive (any number of paths). Many workflows also require that a subset of their activities be repeated, based on similarly evaluated runtime conditions. To capture such requirements regarding the control flow, the workflow net application model is assumed. For a queue of workflows with stochastic activities structured in accordance to this underlying model, an online scheduling policy is proposed, which optimizes the expected response time of the system. It is much more broadly applicable than any competing task graph analysis-based approach, as it captures arbitrary combinations of conditional paths corresponding to both exclusive and non-exclusive choices and of loops with both finite and infinite iterations. At the same time, it is more accurate than competing online algorithms.

### 6.1 Answers to Research Questions

**Offline Scheduling of DAG-reducible Stochastic Workflows**

1. *Given that the general deterministic schedule length minimization is an NP-hard combinatorial problem, how should an additional uncertainty in the problem inputs, i.e., activity run times, be managed by an offline scheduling policy?*

The expected schedule length minimization problem can be solved by an offline scheduler efficiently, even when for this purpose many partial schedules with randomly distributed inputs must be evaluated. An efficient solution is possible...
by employing scalar estimates of the values of the stochastic variables of interest, i.e., a single value for each variable in each computation step. However, this value must be chosen appropriately, under consideration of the specific operators employed in each step, such as job priority determination or evaluation of a partial solution.

2. As the problem of computing the distribution of the schedule length is \#P-complete and given that optimizing a schedule may require the evaluation of many partial solutions, is it possible to devise methods which can estimate partial schedules efficiently? How can the specific features of the optimization problem, e.g., the addition and comparison operators employed in the objective function, guide the choice of the estimation methods?

When comparing activity runtimes with arbitrary, possibly asymmetric distributions, a meaningful estimate of the runtimes is the sum of the expected value and of the positive semi-deviation, which is the partial standard deviation pointing in the same direction as the comparison operator (maximum). By adopting this estimate, two problems are avoided: the reliance on the expected values alone, which may be not representative for a skewed distribution, and incorrect comparisons based on non-directional measures of dispersion, such as the complete standard deviation. At the same time, the positive semi-deviation is easily computable, given a distribution function.

3. If a task is considered for insertion into an idle time slot on a processor and its tentative start and finish times and those of the previously scheduled tasks are stochastic variables, what is a computationally efficient method of estimating the impact of a scheduling decision on the whole solution?

When a task of lower priority is considered for insertion into an idle time slot on a processor, to estimate whether the insertion is likely to cause a significant delay for some of the tasks of higher priority, a local look-ahead mechanism can be employed. In this approach, the insertion of the task of lower priority is simulated. On the basis of the stochastic estimates mentioned above, it is possible to approximate the delay that could be caused (local loss) for any task of higher priority and the time gained (local gain) by the task of lower priority due to insertion. In the specific policy proposed, a trade-off value is computed, which expresses the potential loss with respect to the gain but also takes into account...
the relative priorities of the jobs. To determine the acceptable trade-off value, the policy uses a parameter-driven approach. This conceptually corresponds to a controlled greediness strategy, i.e., the policy is greedier or lazier depending on the parameter value.

4. Given that the run time of an activity that has not been scheduled yet will depend on the speed of the resource (processor or communication channel) it will be scheduled to, what is the best strategy of estimating the resource speed before the choice is made?

In general, when a scheduling policy based on the earliest finish time heuristic is employed, the tasks considered for scheduling are always assigned to the fastest processor available. Thus the average processing time for across all processors available in a heterogeneous platform almost never corresponds to the actual runtime. The assumption of nominal (assignment-independent) processing times during the determination of task priorities is results in a better performance and robustness of the scheduler.

5. In platforms with different processing capacity, i.e., different speed and number of processing units, how meaningful is the ratio of the available processing units to the size of the task graph and how can it be employed to guide the behavior of the scheduler?

An effective approach is to express the processing capacity of the scheduling platform in relation to the size of the task graph, not in absolute terms. This provides a meaningful measure of the processing capacity available with respect to the requirements of the problem. However approximate, the resulting coefficient can be successfully employed as the parameter driving the greediness of the scheduling policy. Experimentally, slightly greedier policies tend to perform better in the stochastic settings investigated, when the combined platform processing capacity is relatively small.

Online Scheduling of DAG-reducible Stochastic Workflows

1. How should the uncertainty in the problem inputs be managed by an online scheduling policy? In particular, given a task considered for scheduling on a processor, if its tentative finish time and the tentative finish times of other tasks
of different, possibly higher priority, are stochastic variables, which methods can estimate the impact of a scheduling decision efficiently?

The response time minimization problem can be solved by an online scheduler efficiently, by employing scalar estimates of the values of the stochastic variables of interest. An online scheduling policy does not insert tasks into actual, bounded idle time slots. However, it can employ a local look-ahead to determine to which extent scheduling a task of lower priority greedily could delay a task of higher priority, which has not yet been scheduled. Conversely, the gain associated with the greedy scheduling decision can be estimated. Based on these estimated values, the final decision can be made. In the proposed policy, the two values, scaled by the relative task priorities, are compared with a threshold value.

2. What methods can improve the accuracy of the estimation of partial schedules in an online scheduler? How can the specific features of the optimization problem guide the choice of these estimation methods?

The accuracy of the estimation of partial schedules in an online scheduler can be improved by employing estimates based on conditional partial moments of the relevant stochastic variables. These are the variables that describe the activities that are under processing when the scheduler performs the estimation.

3. Given the expected response time of the system as a global objective, how should a scheduling policy balance the objectives of expected instance makespan and in-queue waiting time, which may be conflicting for different instances?

An additional local look-ahead subroutine can be employed specifically to minimize the waiting time of new workflow instances. New instances can be started before the termination of the instances currently under processing, when the completion time of the instances of higher priority is unlikely to be affected significantly. In our approach, the addition of this subroutine significantly affects the perceived performance of the policy in most experimental settings.

4. Given a stochastic arrival process of workflow instances, what is a meaningful measure of the different load conditions of the system, resulting from different arrival rates of new workflow instances?

A meaningful measure of the different load conditions of the system, resulting from different arrival rates of new workflow instances, is the ratio of the actual
arrival rate to the critical rate determined with a reference scheduling policy for the system. The critical rate is defined as a rate beyond which the queue will grow indefinitely, if the reference policy is applied to schedule the instances in the queue on the given scheduling platform.

5. If a specific arrival rate is determined with a reference scheduling policy, how can the ratio of the actual arrival rate to the reference rate guide the behavior of the scheduler?

The ratio of the actual arrival rate to the critical reference policy-based rate can effectively guide the behavior of the scheduler, in terms of its greediness. Greedier policies tend to perform better in the stochastic settings investigated, when the relative arrival rate is small. When the opposite is true, lazier policies must be employed in queues with a large number of instances, or potentially infinite instances, to avoid problems with the stability of the queue.

Scheduling of Petri Net-reducible Stochastic Workflows

1. What features of the application model are necessary for realistic workflows that are unavailable in the directed acyclic graph model? Which candidate models have the required expressive power and which criteria should guide the choice of the model?

Realistic workflows require a control flow model that can express conditional and iterative control flow, besides series-parallel execution of activities. Conditional execution corresponding to both exclusive and non-exclusive choice of branches, i.e., activities and partial orders defined on them, need to be supported in a practical workflow model. Some models from business management domain, such as those based on UML or BPMN, have the required expressive power, but are not formally well-defined. Only Petri net-based approaches are at the same time formally well defined and close to the minimal superset of the directed acyclic graph model with the required expressive power and provide a rich tool set of analytic methods.

2. If the control flow admits conditional execution and iterations of activities, which efficient methods can be employed to determine the set of schedulable jobs in a policy that is required to make many such determinations?
If the control flow admits conditional execution and iterations of activities, an efficient method for the determination of the set of schedulable jobs can employ task labelling. In the proposed policy, the approximate set of potentially active tasks is obtained as follows. Upon a choice of a conditional branch, the policy labels as inactive the first tasks in all the discarded branches and all the tasks within those branches which only have inactive-labelled parents. The procedure typically stops at the next OR-join that corresponds to a choice not made yet.

3. How can the task priority determination problem be solved in the presence of conditional branches and loops of activities in a setting with stochastic activities? Which methods can support conditional paths corresponding to both exclusive and non-exclusive choices in the control flow?

In the presence of conditional branches, the problem of estimating the length of multiple alternative paths (before any branch is chosen) can be solved by a method based on weighting the paths of interest in accordance with the prior probabilities assigned to each branch. In our approach, for non-exclusive choices, any combination of choices is considered, i.e., the corresponding set of exclusive choices is made explicit. The exclusive choices represent independent events, whose prior probabilities can be derived, given the probabilities of the non-exclusive choices, which are known. As the path length estimation procedure only employs these scalar values computed locally and requires no further knowledge of the workflow model, the model does not need to be explicitly modified for this purpose.

4. What is an efficient strategy for the problem of estimating the length of a loop containing stochastic activities? How accurate can such a strategy be? Which methods can estimate the lengths of finite and potentially infinite loops?

From the point of view of an activity that is connected to a loop, but outside of it, there exists a formal proof that the loop (finite or not) can be treated as a virtual sequence of activities. These activities are described by a set of supplementary random variables, linked to the original ones by coefficients derived based on the probabilities of exiting the loop, or remaining in it. They also take into account possible multiple exit points of complex loops and their combinations. The overall accuracy depends on the employed estimates of the activity runtimes.
Chapter 6. Conclusions

5. Finally, how can an overall set of length estimation strategies be designed to support combinations of series-parallel, conditional and iterative execution paths?

Given the strategies for the estimation of the length of a set of conditional paths and of paths containing loops outlined above, the classical prioritizing methods in list scheduling can be reformulated within the stochastic workflow net context. In particular, algorithms with a polynomial complexity can be derived from the corresponding heuristics that compute critical paths and dominant sequence, or task levels (bottom and top levels or dynamic levels) in directed acyclic graphs.

The performance and robustness of all the approaches proposed and their component heuristics are discussed in detail in the respective chapters.

6.2 Outlook and Future Work

For the problem of prioritizing jobs in stochastic settings, estimates based on positive partial and conditional partial moments were proposed and applied in this thesis. This is a choice that reflects a global execution or response time minimization objective. It may reflect other objective functions, which utilize the maximum-of comparison operator when the length of execution paths joint in a synchronization point is determined. In other scheduling optimization problems, negative moments could be chosen as estimates, e.g., when reliability or availability aggregation models are considered, which utilize the minimum-of operator at joints. The applicability and the performance of scheduling heuristics based on these estimates should be investigated.

As outlined in Sect. 5.2, the workflow net application model supports all structured control flow patterns, with the exception of those that feature cancellation of a subset of activities. Such workflows, however, are also not uncommon in practice and thus relevant in many actual settings. A workflow net scheduler as the one proposed in this thesis could be extended in order to support this model. The investigation of the additional complexity and possible methods of reducing constitute interesting objects of future study. Similarly, job set determination and prioritizing in unstructured workflows are a challenging avenue of research, given that in such workflows a job can be contained simultaneously in many overlapping substructures of different kinds.
6.2. Outlook and Future Work

All scheduling policies put forward in this thesis can be directly applied to the problems for which they are defined and designed, but they can also support a broader decision process, related to a known stochastic process model and a given processing environment. Let us examine the following scenario. A given deadline imposed over all the workflow instances processed by the available scheduling platform. It translates to a maximum allowed makespan or system response time. In this case, it is possible to determine through simulation how often or by how much accumulated time the deadline is missed. If those metrics deliver an unsatisfactory result, it can also be tested through the same simulation engine, whether increasing the number or speed of the processors solves the problem. However, such a trial-and-error approach is clearly suboptimal. Moreover, depending on the structure of the task graph, it is possible that there would be little or no gain from adding more processors, e.g., if the graph is highly sequential, or, in general, has bottlenecks. In that case, only increasing the speed of some processors could help, but this might not be possible in practice: Clearly, processor speeds cannot be increased arbitrarily (employees, such as office clerks, are a particularly good example). This kind of result would suggest the need for redesigning the workflow: At least some of the bottleneck tasks would have to be made malleable or, in general, processable in parallel, or the workflow schema should be changed. At the time of this writing, there exist approaches able to individuate bottlenecks, but no automated approach which would provide recommendations to redesign workflows to meet prescribed performance criteria. A system capable of such recommendations is an extremely interesting and challenging avenue of future research.
Appendix A

Experiment-Specific Distributions

This appendix outlines specific computation methods for most experimental settings in this thesis, which employ Beta distributions. Depending on the parametrization, a Beta distribution can approximate most continuous probability distributions encountered in practice. The Beta distribution is part of the Pearson family of distributions (Def. A.1 below, following Kendall and Alan [134]), that includes also such distributions as normal, gamma, chi-square and Student’s $t$.

**Definition A.1.** A Pearson distribution with parameters $a, b_0, b_1, b_2 \in \mathbb{R}^+$ is one whose probability distribution function (PDF) $f(x)$ is a solution to the differential equation:

\[
(b_0 + b_1 x + b_2 x^2) df(x) = (x - a) f(x) dx
\]  

(A.1)

The Beta distribution is defined on the basis of the Gamma function (PDF) and of the Beta function (its cumulative distribution function, CDF).

**Definition A.2.** A random variable $X$ has a Beta distribution with parameters $\alpha, \beta \in \mathbb{R}^+$, written $X \sim \text{Beta}(\alpha, \beta)$, if it has a PDF of the form:

\[
f_B(x|\alpha, \beta) = \frac{\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\beta)} x^{\alpha-1}(1 - x)^{\beta-1}
\]  

(A.2)

which corresponds to the CDF of the form:

\[
F_B(x|\alpha, \beta) = \frac{B(x|\alpha, \beta)}{B(\alpha, \beta)}
\]  

(A.3)
where the Gamma function is:
\[
\Gamma(y) = \int_0^\infty t^{y-1}e^{-t}dt
\]  
(A.4)

and the Beta function is:
\[
B(y|\alpha, \beta) = \int_{-\infty}^y t^{\alpha-1}(1-t)^{\beta-1}dt
\]  
(A.5)

with
\[
B(\alpha, \beta) = B(y|\alpha, \beta)|(y = 1)
\]  
(A.6)

In particular, for a Beta distribution with parameters \(\alpha, \beta\), the following holds.

If \(X \sim Beta(\alpha, \beta)\), then the expected value (mean) of \(X\) is given by:
\[
E[X \sim B] = \frac{\alpha}{\alpha + \beta}
\]  
(A.7)

and the first moment about the mean (the variance) of \(X\) is given by:
\[
Var[X \sim B] = \frac{\alpha\beta}{(\alpha + \beta)^2(\alpha + \beta + 1)}
\]  
(A.8)

The following result is due to Winkler et al. [263] (see also Eq. 3.3).

If \(X \sim Beta(\alpha, \beta)\), then the \(n\)-th partial moment of \(X\) about the origin over the interval \((-\infty, z]\) is given by:
\[
E_{-\infty}^{z}[X^n_{\sim B}] = \frac{\Gamma(n + \alpha)\Gamma(\alpha + \beta)}{\Gamma(\alpha)\Gamma(\alpha + \beta)} F_B(z|n + \alpha, \beta)
\]  
(A.9)

The simple Beta distribution is defined (i.e., has a bounded support) over the interval \([0, 1]\). Clearly, most realistic activity times and other QoS attributes may assume values that do not comply with this restriction. This results in the requirement to incorporate scaling, e.g., through the use of the four-parameter Beta distribution, where the two additional parameters \(a, b\) are the support limits of the (scaled) distribution.

**Definition A.3.** A random variable \(X\) has a Beta distribution with four parameters
Appendix A. Experiment-Specific Distributions

\(\alpha, \beta, \in \mathbb{R}^+, a, b \in \mathbb{R}\), written \(X \sim \text{Beta4}(\alpha, \beta, a, b)\), if it has a PDF of the form:

\[f_{B4}(x|\alpha, \beta, a, b) = \frac{f_B(x|\alpha, \beta)}{b-a} = \frac{\Gamma(\alpha + \beta)}{(b-a)\Gamma(\alpha)\Gamma(\beta)}x^{\alpha-1}(1-x)^{\beta-1}\]  \(\text{(A.10)}\)

The following results have been derived for the purpose of this thesis.

If \(X \sim \text{Beta4}(\alpha, \beta, a, b)\), then, by Def. 3.1, the partial expectation of \(X\) over an interval \([y, z]\) is given by:

\[E_y^z[X \sim B4] = \int_y^z xf(x)dx = \frac{\int_y^z f_B(x|\alpha + 1, \beta)dx}{b-a} = \frac{[F_B(x|\alpha + 1, \beta)]_y^z}{b-a}\]  \(\text{(A.11)}\)

and the complete expectation of \(X\) (the marginal mean) is given by:

\[E[X \sim B4] = \mu_{\sim B4} = \frac{\alpha}{(\alpha + \beta)(b-a)}\]  \(\text{(A.12)}\)

If \(X \sim \text{Beta4}(\alpha, \beta, a, b)\), then, by Eq. 3.3, the \(n\)-th partial moment of \(X\) about the origin over the interval \([y, z]\) is given by:

\[E_y^z[X_{\sim B4}^n] = E_y^z\left[\frac{X_{\sim B4}}{(b-a)^n}\right] = \Gamma(n + \alpha)\Gamma(\alpha + \beta)\frac{[F_B(x|n + \alpha, \beta)]_y^z}{\Gamma(\alpha)\Gamma(\alpha + \beta)(b-a)^n}\]  \(\text{(A.13)}\)

and, by Eq. 3.6, the \(n\)-th partial moment of \(X\) about the point \(x_0\) over the interval \([y, z]\) is given by:

\[E_y^z[(X_{\sim B4} - x_0)^n] = \sum_{i=0}^{n} \binom{n}{i} (-x_0)^i E_y^z[X_{\sim B4}^{n-i}]\]  \(\text{(A.14)}\)

In particular, the partial variance (second moment about the mean) is given by the following formula.

If \(X \sim \text{Beta4}(\alpha, \beta, a, b)\) and if \(\mu_{\sim B4}\) is the marginal mean of \(X\), then the partial variance of \(X\) over an interval \([y, z]\) is given by:

\[Var_y^z[X_{\sim B4}] = 1(-\mu_{\sim B4})^0 E_y^z[X_{\sim B4}^{2-0}] + 2(-\mu_{\sim B4}) E_y^z[X_{\sim B4}^{2-1}] + 1(-\mu_{\sim B4})^2 E_y^z[X_{\sim B4}^{2-2}]\]  \(\text{(A.15)}\)
Therefore:

\[ \text{Var}_{y}[X_{\sim B4}] = E_{y}^{2}[X^{2}_{\sim B4}] - 2\mu_{\sim B4}E_{y}^{1}[X_{\sim B4}] + \mu_{\sim B4}^{2} E_{y}^{0}[X_{0\sim B4}] \]  

(A.16)

which implies:

\[ \text{Var}_{y}[X_{\sim B4}] = E_{y}^{2}[X^{2}_{\sim B4}] - 2\mu_{\sim B4}E_{y}^{1}[X_{\sim B4}] + \mu_{\sim B4}^{2} \]  

(A.17)

and finally:

\[ \text{Var}_{y}[X_{\sim B4}] = \left[ F_{B}(x|\alpha + 2, \beta) \right]_{y}^{z} - 2\mu_{\sim B4} \left[ F_{B}(x|\alpha + 1, \beta) \right]_{y}^{z} + \mu_{\sim B4}^{2} \]  

(A.18)

Therefore, the positive semi-variance of \( X \) is given by:

\[ \text{Var}_{+}[X_{\sim B4}] = \frac{[F_{B}(x|\alpha + 2, \beta)]_{\mu_{\sim B4}}^{\infty}}{(b - a)^{2}} - 2\mu_{\sim B4} \frac{[F_{B}(x|\alpha + 1, \beta)]_{\mu_{\sim B4}}^{\infty}}{b - a} + \mu_{\sim B4}^{2} \]  

(A.19)

In general, because the support limits \( a, b \) of the distribution are constant with respect to the free variable \( x \) in the distribution functions, they can be moved freely, disregarding the integration (or derivation) operators. The \( n \)-th moment of the four-parameter Beta distribution is therefore, naturally, scaled by \( (b - a)^{-n} \). For \( b = 1, a = 0 \) we naturally obtain the two-parameter Beta distribution formulas.

For a condition of the type \( X \in A \), the conditional moments are all scaled by the integral \( \int_{A} dF(x) = [f(x)]_{A} \), according to Theorem 3. This integral becomes \( [f(x)]_{z}^{\infty} \) for the condition of type \( X \geq z \) used for the estimates of already running activities by the online scheduler in Chapter 4 and beyond.

In the experimental evaluation (Sections 3.6, 4.5, 5.6), for each activity, a four-parameter Beta distribution \( X \sim Beta4(\alpha, \beta, a, b) \) is extrapolated based on the prescribed values of its mean \( E[X] = \mu_{\sim B4} \) and normalized standard deviation \( \frac{\text{Var}[X]}{E[X]} = n_{\sim B4} \). The support limits \( a > 0, b > \mu_{\sim B4} \) are assumed to be a constant-factor fraction of \( \mu_{\sim B4} \) and multiplicity of \( \mu_{\sim B4} \), respectively. The shape parameters \( a > 0, b > 0 \) are derived by transforming the formulas in Eq. A.7 A.8 We obtain:

\[ a = a_{mul}\mu_{\sim B4} \]  

(A.20)
where $a_{mul}, b_{mul} \in \mathbb{R}^+$ with $a_{mul} < 1, b_{mul} > 1$ and

\[
\alpha = \frac{\mu_{\sim B4}}{n_{\sim B4}} (\mu_{\sim B4} + 1 - n_{\sim B4}) \quad \text{(A.22)}
\]

\[
\beta = \alpha \frac{(1 - \mu_{\sim B4})}{\mu_{\sim B4}} = \frac{1 - \mu_{\sim B4}}{n_{\sim B4}} (\mu_{\sim B4} + 1 - n_{\sim B4}) \quad \text{(A.23)}
\]

where $n_{\sim B4} \in \mathbb{R}^+$ and it must hold that $n_{\sim B4} < \mu_{\sim B4} + 1$. 
Bibliography


[32] L. F. Bittencourt and E. R. Madeira. Towards the scheduling of multiple work- 


[34] J. Błażewicz, K. H. Ecker, G. Schmidt, and J. Węglarz. *Scheduling in computer 

[35] J. Błażewicz, J. K. Lenstra, and A. Kan. Scheduling subject to resource con-
24, 1983.

1974.

[37] S. H. Bokhari. A shortest tree algorithm for optimal assignments across space 
and time in a distributed processor system. *Software Engineering, IEEE Trans-

[38] L. Bölöni and D. C. Marinescu. Robust scheduling of metaprograms. *Journal 

constrained project scheduling: Notation, classification, models, and methods*. 


service times to minimize the expected flow time or makespan. *Journal of the 

of the eighteenth annual ACM symposium on Parallelism in algorithms and ar-

on analytical performance and QoS in cloud computing environments. In *Pro-
cedings of the International Conference on Parallel Processing*, pages 295–
304. IEEE, 2011.


