

## **NOTE**

This online version of the thesis may have different page formatting and pagination from the paper copy held in the Swinburne Library.

# **Scheduling Algorithms for Instance-Intensive Cloud Workflows**

by

**Ke Liu**

**B.Sci. (Nanjing University of Science and Technology)**

**M.Eng. (Huazhong University of Science and Technology)**

A thesis submitted to

**CS3 – Centre for Complex Software Systems and Services**

**Faculty of Information and Communication Technologies**

**Swinburne University of Technology**

for the degree of

**Doctor of Philosophy**

**June, 2009**

*To my parents and sister*

# 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 in the text of the thesis. 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.*

**Ke Liu**

**June, 2009**

# Acknowledgements

I would like to express my sincere gratitude to the following people and institutions for whose help I will be forever grateful.

First of all, I sincerely give the deepest gratitude to my coordinating supervisors, Professor Yun Yang and Professor Hai Jin and associate supervisor Doctor Jinjun Chen, for their seasoned supervision and advice throughout the life cycle of my PhD studies, and for their careful and patient proofreadings and modifications of this thesis. Without their consistent support, I would never be able to finish this manuscript.

I want to give my thanks to all the staff members, research students and research assistants in Centre for Complex Software Systems and Services (CS3) for all the help they have given me during my PhD study time. Thanks to the members of the Workflow Team, namely, Qiang He, Xiao Liu and Dong Yuan, for giving advice and cooperation on my research and companionship in sports and daily life which made my PhD study period more enjoyable. I am very grateful for Dr Antony Tang's help in understanding the bank cheque processing process. Special thanks also go to Bryce Gibson for proofreading the English of this Thesis.

I also want to thank Swinburne University of Technology and the Faculty of Information and Communication Technologies for offering me a full scholarship throughout my doctoral program. Thanks to CS3 for research publication funding support and supporting me to attend local and international conferences.

Last but not least, I deeply thank my parents and sister for their support, understanding, love, encouragement and sacrifice that always helped me to overcome any difficulty that was encountered during my life.

# Abstract

This thesis focuses on algorithms for scheduling instance-intensive cloud workflows in different cloud computing environments built on different system infrastructures. Instance-intensive cloud workflows are workflows with a huge number of workflow instances (hence instance intensive) running on a cloud computing environment (hence cloud workflows).

First of all, the prosperity of e-business relies on the ability to process instance-intensive workflows. A typical example of instance-intensive workflows is the bank cheque processing scenario, where millions of cheque-processing transactions need to be processed concurrently each day, while each of them is a rather simple workflow with only a few steps.

Meanwhile, with the promotion of the world's leading companies, cloud computing has become an area of increasing interest. Cloud computing has many unique advantages, such as low cost, scalability, reliability and fault-tolerance, which can effectively facilitate the execution of workflows.

Moreover, cloud computing environments can be built on different system infrastructures. For instance, it can be built on physically collocated grids (grid-based), or geographically distributed services (service-based). In a commercial cloud computing infrastructure with a “pay per use” context (business-based) the execution cost should be considered as well as the execution time. Each of these different infrastructures demands a different workflow scheduling algorithm.

Therefore, in this research, we propose three corresponding algorithms for scheduling instance-intensive workflows on grid-based, service-based and business-based cloud computing environments respectively, namely, TMS (Throughput Maximisation Strategy) for a grid-based cloud computing environment, MMA (Min-Min-Average) for a service-based cloud computing environment and CTC (Compromised-Time-Cost) for a business-based cloud computing environment.

TMS consists of two algorithms in order to adapt to a grid-based cloud computing environment. The first algorithm is to maximise the overall throughput by pursuing overall load balance at the instance level, while the second algorithm strives to further maximise the throughput by increasing the utilisation rate of resources within each local autonomous group at the task level. Through these efforts, an increase in overall throughput can be achieved.

MMA aims at increasing the overall throughput in a service-based cloud computing environment. It establishes a fundamental infrastructure that can provide nearest neighbours to decrease the communication time significantly. Moreover, due to the importance of both execution time and transmission time, it uses a different strategy to the original Min-Min algorithm in order to improve the utilisation rate of both the execution unit and transmission unit of the workflow execution engine in order to increase the overall throughput.

Designed for business cloud workflows, the CTC algorithm is targeted to compromise the time and cost through the scheduling process. For different user requirements, this algorithm can be further divided into two sub-algorithms: the CTC-MC (Compromised-Time-Cost algorithm Minimising execution Cost) algorithm, which minimises the execution cost within user designated deadline, and the CTC-MT (Compromised-Time-Cost algorithm Minimising execution Time) algorithm, which minimises the execution time within user designated budget. However, both algorithms allow the user to request an acceptable compromise between execution time and execution cost on the fly.

The simulations performed on Swinburne Workflow Test Environment (SWTE) demonstrate that all our workflow scheduling algorithms have better performance than their counterparts which are scheduling algorithms currently implemented in most popular workflow management systems.

# The Author's Publications

## Journals

- [1] K. Liu., J. Chen, Y. Yang and H. Jin, A Throughput Maximization Strategy for Scheduling Transaction Intensive Workflows on SwinDeW-G, *Concurrency and Computation: Practice and Experience*, 20(15), 1807-1820, October 2008.

## Journals (Submitted)

- [2] K. Liu, J. Chen, H. Jin and Y. Yang, Min-Min-Average: A Scheduling Algorithm in SwinDeW-G for Instance-intensive Grid Workflows Involving Considerable Communication Overheads, Submitted to *Journal of Parallel and Distributed Computing*.
- [3] K. Liu, Y. Yang, J. Chen, X. Liu, D. Yuan and H. Jin, A Compromised-Time-Cost Scheduling Algorithm in SwinDeW-C for Instance-intensive Cost-Constrained Workflows on Cloud Computing Platform, submitted to *International Journal of High Performance Computing Applications*.

## Conferences

- [4] Y. Yang, K. Liu, J. Chen, J. Lignier, and H. Jin, Peer-to-Peer Based Grid Workflow Runtime Environment of SwinDeW-G, *Proc. of the 3rd IEEE International Conference on e-Science and Grid Computing*, 51-58, Bangalore, India, December 2007.
- [5] Y. Yang, K. Liu, J. Chen, X. Liu, D. Yuan and H. Jin, An Algorithm in SwinDeW-C for Scheduling Transaction-Intensive Cost-Constrained Cloud Workflows, *Proc. of 4th IEEE International Conference on e-Science*, 374-375, Indianapolis, USA, December 2008.

- [6] X. Liu, J. Chen, K. Liu and Y. Yang, Forecasting Duration Intervals of Scientific Workflow Activities based on Time-Series Patterns, Proc. of 4th IEEE International Conference on e-Science, Indianapolis, 23-30, Indianapolis, USA, December 2008.
- [7] K. Liu, J. Chen, H. Jin and Y. Yang, A Min-Min Average Algorithm for Scheduling Transaction-Intensive Grid Workflows, Proc. of 7th Australasian Symposium on Grid Computing and e-Research, 41-48, Wellington, New Zealand, January 2009.

# Table of Contents

<b>Scheduling Algorithms for Instance-Intensive Cloud Workflows.....</b>	<b>i</b>
<b>Declaration .....</b>	<b>ii</b>
<b>Acknowledgements .....</b>	<b>iii</b>
<b>Abstract .....</b>	<b>iv</b>
<b>The Author’s Publications .....</b>	<b>vi</b>
<b>Table of Contents .....</b>	<b>viii</b>
<b>List of Figures.....</b>	<b>xiii</b>
<b>List of Tables .....</b>	<b>xv</b>
<b>Chapter 1 Introduction.....</b>	<b>1</b>
1.1 Introduction to the Work.....	2
1.2 Key Issues of this Research .....	3
1.2.1 Design of Best-effort Scheduling Algorithms for Instance-intensive Cloud Workflows .....	3
1.2.1.1 In the Case of Grid-based Cloud Computing Environment .....	4
1.2.1.2 In the Case of Service-based Cloud Computing Environment.....	4
1.2.2 Design of Scheduling Algorithms for Cost-constrained Instance-intensive Cloud Workflows .....	4
1.3 Overview of this Thesis .....	5
<b>Chapter 2 Literature Review and Requirements Analysis .....</b>	<b>6</b>
2.1 Introduction to Computing Platforms .....	7
2.1.1 Introduction to Grid Computing.....	7
2.1.1.1 Definition of Grid Computing .....	7
2.1.1.2 Features of Grid Computing .....	7
2.1.1.3 Architecture of Grid .....	8
2.1.2 Introduction to Cloud Computing .....	9
2.1.2.1 Definition of Cloud Computing.....	9
2.1.2.2 Features of Cloud Computing.....	10
2.1.2.3 Architecture of Cloud .....	11

2.1.3	Relationship of Cloud Computing and Grid Computing .....	12
2.2	Introduction to Workflow and Workflow Management Systems .....	14
2.2.1	Definition of Workflow .....	14
2.2.2	Architecture of Workflow Management Systems.....	14
2.2.3	Existing Workflow Management Systems.....	16
2.2.3.1	Conventional Client-Server Workflow Management Systems.....	16
2.2.3.2	Peer-to-peer Workflow Management Systems.....	16
2.2.3.3	Grid Workflow Management Systems .....	16
2.2.3.4	Cloud Workflow Management Systems.....	17
2.2.3.5	Our Own Workflow Management Systems for Grid and Cloud .....	17
2.3	Introduction to Workflow Scheduling Algorithms .....	18
2.3.1	Best-effort Based Workflow Scheduling Algorithms .....	19
2.3.1.1	A Hybrid Heuristic.....	19
2.3.1.2	TANH.....	19
2.3.1.3	Dynamic Critical Path for Grids (DCP-G) .....	20
2.3.1.4	Fast Critical Path (FCP) Algorithm .....	20
2.3.1.5	Adaptive Generalised Scheduler (AGS).....	21
2.3.1.6	Workflow Mapping Mechanism (WMM) .....	21
2.3.1.7	Adaptive Workflow Splitting (AWS) Algorithm .....	22
2.3.1.8	Adaptive Scheduling Algorithm (ASA) .....	23
2.3.1.9	Heterogeneous Earliest-Finish-Time Algorithm (HEFT).....	23
2.3.1.10	Greedy Randomised Adaptive Search Procedure (GRASP) .....	24
2.3.1.11	Simulated Annealing (SA) Algorithm.....	24
2.3.1.12	Genetic Algorithms (GA).....	25
2.3.1.13	Myopic Algorithm .....	25
2.3.1.14	Min-Min, Max-Min and Sufferage Heuristic .....	26
2.3.2	QoS-constraint Based Workflow Scheduling Algorithms .....	27
2.3.2.1	Back-tracking Algorithm.....	27
2.3.2.2	Loss and Gain Approach .....	27
2.3.2.3	Genetic Algorithm Approach .....	28
2.3.2.4	Improved Genetic Algorithm.....	28
2.3.2.5	Ant Colony System (ACS) Algorithm.....	29
2.3.2.6	Deadline Distribution Algorithm.....	30
2.4	Problem and Requirements Analysis .....	31
2.4.1	Example Scenario of Bank Cheque Processing .....	31

2.4.2	Problem Analysis .....	31
2.4.3	Requirements Analysis .....	32
2.5	Summary .....	33
<b>Chapter 3 Throughput Maximisation Strategy for Scheduling Instance-intensive</b>		
<b>Cloud Workflows .....</b>		
		<b>34</b>
3.1	Introduction.....	34
3.2	Requirements Analysis and Overall Solution .....	35
3.2.1	Requirements Analysis .....	35
3.2.2	Overall Solution .....	36
3.3	Scheduling Infrastructure.....	36
3.4	Opposite Average Load (OAL) and Extended Min-min (EMM) Algorithms .....	38
3.4.1	OAL (Opposite Average Load) Scheduling Algorithm .....	38
3.4.2	EMM (Extended Min-Min) Scheduling Algorithm .....	40
3.4.3	Complexity Analysis.....	42
3.4.3.1	Computation Complexity of OAL Algorithm .....	42
3.4.3.2	Computation Complexity of EMM Algorithm .....	43
3.5	Example for Demonstration.....	43
3.5.1	Example for Demonstration of the OAL Algorithm .....	43
3.5.1.1	Scheduling Process of the Original Gossip Algorithm.....	44
3.5.1.2	Scheduling Process of the OAL Algorithm .....	45
3.5.2	Example for Demonstration of the EMM Algorithm.....	46
3.6	Comparison and Simulation.....	47
3.6.1	Simulation Environment .....	47
3.6.2	Criteria for Comparison .....	49
3.6.2.1	Criteria for Comparison between the Original Gossip Algorithm and the OAL Algorithm .....	49
3.6.2.2	Criteria for Comparison between the Original Min-Min Algorithm and EMM Algorithm .....	50
3.6.3	Simulation Process .....	50
3.6.4	Results and Analysis .....	51
3.7	Summary .....	54
<b>Chapter 4 Min-Min Average Algorithm for Scheduling Instance-intensive Cloud</b>		
<b>Workflows Involving Communication Overhead .....</b>		
		<b>55</b>
4.1	Introduction.....	55
4.2	Requirements Analysis and Overall Solution .....	56
4.2.1	Requirements Analysis .....	56

4.2.2	Overall Solution .....	57
4.3	Scheduling Infrastructure.....	57
4.3.1	Service cloud.....	58
4.3.2	Ordinary Node.....	58
4.3.3	Monitor Node .....	60
4.3.4	Node Maintenance .....	60
4.3.5	Node Search .....	61
4.4	MMA Scheduling Algorithm.....	61
4.4.1	Terms Used in MMA Algorithm.....	62
4.4.2	Overview of MMA.....	63
4.4.3	Details of MMA.....	65
4.4.4	Complexity Analysis.....	68
4.5	Example for Demonstration.....	69
4.6	Comparison and Simulation.....	71
4.6.1	Simulation Environment .....	71
4.6.2	Criteria for Comparison .....	73
4.6.3	Simulation Process.....	73
4.6.4	Results and Analysis .....	74
4.7	Summary .....	76
<b>Chapter 5 Compromised-Time-Cost Algorithm for Cost-constrained Cloud</b>		
<b>Workflows</b>	.....	<b>77</b>
5.1	Introduction.....	77
5.2	Requirements Analysis and Overall Solution .....	78
5.2.1	Requirements Analysis .....	78
5.2.2	Overall Solution .....	79
5.3	Scheduling Infrastructure.....	80
5.3.1	Service Cloud.....	80
5.3.2	Cloud Workflow Execution Agent (CWEA) .....	81
5.3.3	Cloud Service Catalogue.....	81
5.3.4	User Interface .....	81
5.4	CTC Scheduling Algorithm.....	82
5.4.1	CTC-MC Scheduling Algorithm.....	83
5.4.2	CTC-MT Scheduling Algorithm .....	89
5.4.3	Complexity Analysis.....	90
5.4.3.1	Computation Complexity of CTC-MC Algorithm .....	90
5.4.3.2	Computation Complexity of CTC-MT Algorithm .....	90

5.5	Example for Demonstration.....	91
5.5.1	Scheduling Procedure of Deadline-MDP algorithm .....	92
5.5.2	Scheduling Procedure of CTC-MC Algorithm .....	96
5.6	Simulation and Comparison.....	100
5.6.1	Simulation Environment .....	100
5.6.2	Criteria for Comparison .....	102
5.6.3	Simulation Process.....	102
5.6.4	Results and Analysis .....	103
5.7	Summary .....	104
<b>Chapter 6</b>	<b>Discussion.....</b>	<b>106</b>
6.1	Improvements to OAL Algorithm in TMS Strategy .....	106
6.1.1	Deficiency of OAL Algorithm in TMS Strategy .....	106
6.1.2	Suggestion for Improvement.....	107
6.2	Improvement of MMA Algorithm .....	108
6.2.1	Deficiency of MMA Algorithm .....	108
6.2.2	Suggestion for Improvement.....	109
6.3	Improvement of CTC Algorithm .....	111
6.3.1	Deficiency of CTC Algorithm .....	111
6.3.2	Suggestion for Improvement.....	112
6.4	Summary .....	113
<b>Chapter 7</b>	<b>Conclusions and Future Work.....</b>	<b>114</b>
7.1	Summary of this Thesis .....	114
7.2	Contributions of this Thesis .....	116
7.3	Future work.....	119
<b>Bibliography</b>	<b>.....</b>	<b>121</b>

# List of Figures

Figure 2.1 Grid Architecture [FZRL2008] .....	8
Figure 2.2 Cloud Architecture [FZRL2008].....	12
Figure 2.3 Grids and Clouds [FZRL2008] .....	13
Figure 2.4 WfMC’s Workflow Reference Model.....	15
Figure 3.1 Scheduling Infrastructure of SwinDeW-Cg .....	37
Figure 3.2 Data Structure Used in EMM algorithm .....	40
Figure 3.3 Simple Example of Mesh .....	44
Figure 3.4 Scheduling Result of Original Gossip Algorithm .....	45
Figure 3.5 Scheduling Result of OAL Algorithm .....	46
Figure 3.6 Example of Scheduling Results of Original and Extended Min-Min Algorithms.....	47
Figure 3.7 Swinburne Workflow Test Environment (SWTE).....	48
Figure 3.8. Swinburne Decentralised Workflow for Cloud based on Grid (SwinDeW-Cg) .....	49
Figure 3.9 Variance Deviation of Load Values of Original Gossip and OAL Algorithms .....	51
Figure 3.10 Mean Execution Time of Original Gossip and OAL Algorithms .....	52
Figure 3.11 Resource Utilisation Rate of Original Gossip and OAL Algorithms .....	52
Figure 3.12 Overall Throughput of Original Gossip and OAL Algorithms .....	53
Figure 3.13 Overall Advantage of Proposed Throughput Maximisation Strategy (TMS) .....	53
Figure 4.1 Scheduling Infrastructure of MMA .....	58
Figure 4.2 Data Structure of Ordinary Node .....	59
Figure 4.3 Data Structure of Monitor Node.....	60
Figure 4.4 Scheduling Results of Min-Min Algorithm (a) and MMA Algorithm (b)....	70

Figure 4.5 Swinburne Decentralised Workflow for Cloud based on Service (SwinDeW-Cs) .....	72
Figure 4.6(a): Execution Unit Utilisation Rate of Min-Min and MMA Algorithms .....	74
Figure 4.6(b): Transmission Unit Utilisation Rate of Min-Min and MMA Algorithms .....	74
Figure 4.7: Mean Execution Time of Min-Min and MMA Algorithms .....	75
Figure 4.8: Overall Throughput of Min-Min and MMA Algorithms .....	75
Figure 5.1: Architecture of SwinDeW-Cb .....	80
Figure 5.2: Stream-pipe Mode Sub-deadline Distribution .....	85
Figure 5.3 Example of Time-Cost Relationship Graph .....	88
Figure 5.4 Comparison of Deadline Distributions .....	91
Figure 5.5 Swinburne Decentralised Workflow for Cloud based on Business Model (SwinDeW-Cb) .....	101
Figure 5.6 Comparison on Mean Execution Cost .....	103
Figure 5.7 Comparison on Mean Execution Time .....	104
Figure 6.1 Area for OAL Value Calculation .....	107
Figure 6.2 Example of Resource Clustering .....	108
Figure 6.3 Example of Resource Type Affinity Matrix .....	109
Figure 6.4 An Example of BEA Procedure .....	110
Figure 6.5 Dynamic Adjusting at Run Time .....	112

# List of Tables

Table 3.1 OAL Value Table .....	44
Table 4.1: Pseudo Code of MMA Algorithm .....	63
Table 5.1: Overview of CTC-MC Algorithm .....	83
Table 5.2. Service Processing Speeds and Corresponding Prices for Task Execution...	86
Table 5.3. Service Bandwidths and Corresponding Prices for Data Transmission .....	86

# Chapter 1

## Introduction

This thesis addresses the scheduling algorithms for instance-intensive cloud workflows. Instance-intensive cloud workflows are workflows with a huge number of workflow instances (hence instance intensive) enabled on a cloud computing environment (hence cloud workflows). Unlike complex scientific workflows on which most existing workflow scheduling algorithms focus, the main characteristic of instance-intensive workflows is a huge number of potentially relatively simple concurrent instances. A typical example of these workflows is the bank cheque processing scenario, in which there are millions of concurrent cheque-processing transactions per day, while each of them is a rather simple workflow with only a few steps.

This distinguishing characteristic of instance-intensive workflows makes most existing workflow scheduling algorithms unsuitable for scheduling instance-intensive workflows. Thus this thesis focuses on designing a set of new scheduling algorithms for these workflows based on different cloud infrastructures. For example, workflows that are executed on the grid-based cloud computing infrastructure and service-based cloud computing infrastructure should be scheduled with different strategies. Moreover, if the cost needs to be considered, how to compromise the cost with execution time (makespan) also becomes an important issue. As a result, this thesis proposes different scheduling algorithms for instance-intensive workflows in different circumstances.

## 1.1 Introduction to the Work

By definition, a workflow is *the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules to achieve, or contribute to, an overall business goal* [WfMC99].

Also by definition, a cloud computing environment has *a large pool of easily usable and accessible virtualised resources (such as hardware, development platforms and/or services). These resources can be dynamically re-configured to adjust to a variable load (scale), allowing also for an optimum resource utilisation. This pool of resources is typically exploited by a pay-per-use model in which guarantees are offered by the Infrastructure Provider by means of customised SLAs (service level agreements)* [VRCL2008].

In this thesis, the research objective is the scheduling algorithms for instance-intensive cloud workflows. Literally speaking, cloud workflows are workflows which are executed on a cloud computing environment in order to utilise the advantages that cloud computing provides to facilitate the execution of workflows. These advantages include the following aspects:

- Moving workflows to a cloud computing environment enables the utilisation of various cloud services to facilitate workflow execution.
- In contrast to dedicated resources in grid, the resources in cloud are shared and provided to users by following the "on-demand" feature, which means the expenditure on hardware for workflow execution can be saved.
- The "user-centric" model in cloud makes workflow execution more user-friendly thus increases user satisfaction.
- In contrast to the project-oriented business mode in grid, the "pay as you go" business mode in cloud can save execution cost of workflows, for workflow execution is usually uncertain due to practical execution path.

For these reasons this paper dedicates to research on the scheduling algorithms for instance-intensive cloud workflows. Instance-intensive cloud workflows are *workflows with a huge number of (hence instance-intensive) concurrent workflow instances, usually much simpler than those complex scientific workflows, enabled on a cloud computing environment (hence cloud workflows)*. Workflow scheduling is a process that maps and manages the execution of inter-dependent tasks on the distributed resources [YB2007]. It allocates suitable resources to workflow tasks so that the execution can be completed to satisfy objective functions imposed by users. How to schedule instance-intensive workflows in a cloud computing environment efficiently thus becomes an important issue. Our work is to design a set of new scheduling algorithms for these workflows.

## **1.2 Key Issues of this Research**

Workflows have been a research area for several decades while cloud computing has only recently emerged as a computing paradigm which can be utilised for workflow execution. Particularly, in our research, we focus on designing new scheduling algorithms for instance-intensive workflows in the cloud computing environment. The corresponding key issues are addressed in this section.

### **1.2.1 Design of Best-effort Scheduling Algorithms for Instance-intensive Cloud Workflows**

Best-effort scheduling algorithms are needed for instance-intensive cloud workflows by nature, because the scheduling of these workflows is generally concerned more with the overall throughput. This is in order to complete as many workflow instances in as short a time as possible. Thus the first key objective of this thesis is to design a new best-effort scheduling algorithm for instance-intensive cloud workflows.

Moreover, the infrastructure of cloud computing environment varies, which suggests to us that different algorithms are needed for different cloud infrastructures. The two most common cases are as follows.

### **1.2.1.1 In the Case of Grid-based Cloud Computing Environment**

In the first common case, because clouds and grids share a lot commonality in their vision, architecture and technology, it is possible for clouds to be implemented over existing grid technologies [FZRL2008].

In this case, the cloud computing environment is composed of grids via the Internet where the communication cost within each physically collocated grid is usually negligible. In this circumstance, scheduling algorithms should balance the load among grids first and then maximise the utilisation of resources in each grid to increase the overall throughput.

### **1.2.1.2 In the Case of Service-based Cloud Computing Environment**

In the second common case, the cloud computing infrastructure can also be built on services all over the world with different levels of virtualisation technologies [Bra2008]. This means that the fundamental infrastructure of cloud computing could also be service-based. Here we only discuss the case of free services where overall performance is the top priority concern and cost is not a consideration. The case of charged services will be discussed in Section 1.2.2.

In this case, the cloud computing environment is composed of services scattered on the Internet where the communication overheads can no longer be ignored. In this circumstance, dynamic adaptation to network bandwidth for communication should be reflected in the scheduling algorithms.

## **1.2.2 Design of Scheduling Algorithms for Cost-constrained Instance-intensive Cloud Workflows**

This part of the thesis focuses on a new scheduling algorithm for cost-constrained instance-intensive workflows in a commercial cloud computing environment. Unlike the scenario in Section 1.2.1 where resources are assumed free (because of the possession of the resources or using free services on the Internet), in a commercial cloud computing environment, customers generally do not own the infrastructure, they merely access or rent, so they can avoid capital expenditure and consume resources as a service [Gee2008]. As users are normally sensitive to execution cost, the cost would be another

major concern for cloud workflow applications in addition to execution time. Thus we need to design scheduling algorithms for these cost-constrained instance-intensive cloud workflows.

### **1.3 Overview of this Thesis**

This chapter mainly points out the key issues of scheduling instance-intensive cloud workflows. In the next chapters, Chapter 2 gives a literature review of the background, analyses the problems of existing scheduling algorithms and summarises the requirements for the new algorithms. Chapter 3 proposes a scheduling strategy for instance-intensive workflow on grid-based cloud computing platform, Chapter 4 presents a scheduling algorithm for instance-intensive workflow on service-based cloud computing platform, and Chapter 5 designs a scheduling algorithm for instance-intensive workflows on business-based cloud computing platforms. Chapter 6 identifies some of the deficiencies of this work and gives some suggestions for improvements respectively. Finally, Chapter 7 summarises the thesis and points out the future work.

## **Chapter 2**

# **Literature Review and Requirements Analysis**

In this chapter, we first give an introduction to background of the research issue, then analyse the problems of existing scheduling algorithms when dealing with this particular research scenario, and finally summarise the requirements of the new scheduling algorithms for cloud workflows.

This chapter is organised as follows. A detailed introduction of computing platforms is given in Section 2.1. In particular, Sections 2.1.1 and 2.1.2 introduce grid computing and cloud computing respectively, followed by a comparison of cloud computing and grid computing in Section 2.1.3. Section 2.2 introduces the fundamentals of workflow and workflow management systems. Specifically, Section 2.2.1 introduces the definition of workflow, and Section 2.2.2 introduces the architecture of workflow management systems. In Section 2.2.3, representative workflow management systems (WfMS), namely, client-server workflow management systems, peer-to-peer workflow management systems, grid workflow management systems and cloud workflow management systems are introduced as well as our own workflow management systems. In Section 2.3, the fundamentals of workflow scheduling algorithms are introduced. As a consequence of the classification of workflow scheduling algorithms in this section, Section 2.3.1 and Section 2.3.2 introduce best-effort based scheduling algorithms and QoS-constraint based (of which the most important constraint is the cost constraint) scheduling algorithms respectively. Section 2.4 discusses the problems of existing algorithms when scheduling instance-intensive cloud workflows and presents the

requirements of new algorithms. Finally in Section 2.5, we summarise the work of the chapter.

## **2.1 Introduction to Computing Platforms**

In this section, we introduce the computing platforms related to our work, namely, the grid computing and cloud computing environments.

### **2.1.1 Introduction to Grid Computing**

#### **2.1.1.1 Definition of Grid Computing**

Ian Foster, who is considered “the father of the Grid”, defines a grid as “*a system that coordinates resources which are not subject to centralised control, using standard, open, general-purpose protocols and interfaces to deliver nontrivial qualities of service*” [Fos2002].

#### **2.1.1.2 Features of Grid Computing**

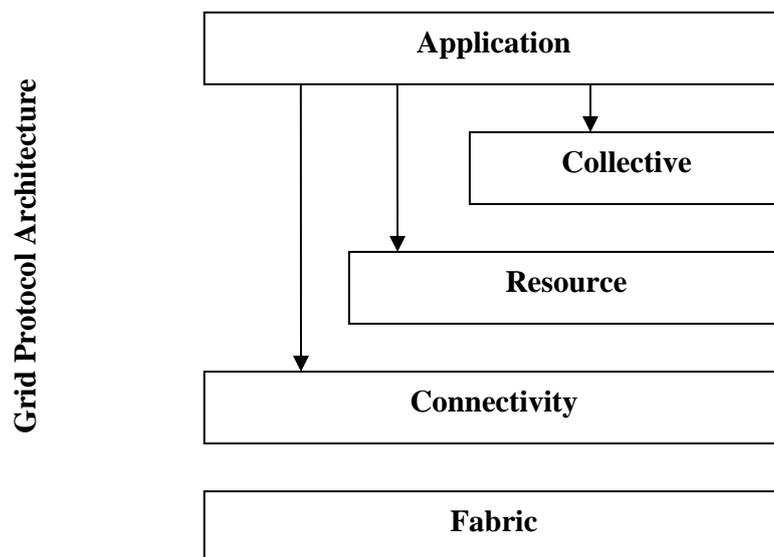
According to IBM [FBAK+2003], grid computing has the following features:

- Grid computing can exploit underutilised resources and run existing applications on them.
- The potential for massive parallel CPU capacity is one of the most attractive features of a grid.
- Grid computing can enable and simplify collaboration among a wider audience.
- In addition to CPU and storage resources, a grid can provide access to increased quantities of other resources and to special equipment, software, licenses, and other services.
- A grid can merge a large number of resources contributed by individual machines into a greater total virtual resource.

- Reliability is achieved via inexpensive and geographically dispersed resources in a grid.
- The goal to virtualise the resources on the grid and more uniformly handle heterogeneous systems will create new opportunities to better manage a larger, more dispersed IT infrastructure.

### 2.1.1.3 Architecture of Grid

Grids provide protocols and services at five different layers as identified in the Grid protocol architecture (see Figure 2.1) [FZRL2008].



**Figure 2.1 Grid Architecture [FZRL2008]**

At the fabric layer, Grids provide access to different resource types such as computing power; storage and networking resources; and code repositories. This is by no means the limit of resources that can be supplied. Grids usually rely on existing fabric components, for instance, local resource managers (e.g. PBS [BHKL+2000], Condor [TTL2005], etc). General-purpose components such as GARA (general architecture for advanced reservation) [FKLL+1999], and specialised resource management services such as Falkon [RZDF+2007].

The connectivity layer defines core communication and authentication protocols for easy and secure network transactions. The GSI (Grid Security Infrastructure) [Glo2005] protocol underlies every Grid transaction.

The resource layer defines protocols for the publication, discovery, negotiation, monitoring, accounting and payment of sharing operations on individual resources. The GRAM (Grid Resource Access and Management) [FK1997] protocol is used for allocation of computational resources and for monitoring and control of computation on those resources, and GridFTP [ABBC+2002] for data access and high-speed data transfer.

The collective layer captures interactions across collections of resources, directory services such as MDS (Monitoring and Discovery Service) [SRPM+2006] that allows for the monitoring and discovery of VO (Virtual Organisation) resources. Condor-G [FTFL+2002] and Nimrod-G [BAG2000] are examples of co-allocating, scheduling and brokering services, MPICH [KTF2003] for Grid enabled programming systems, and CAS (community authorisation service) [FKPT+2003] for global resource policies.

The application layer comprises whatever user applications built on top of the above protocols and APIs, and operate in VO environments. Two examples are grid workflow management systems, and grid portals (e.g. the QuarkNet e-learning environment [ZWFV+2005], the National Virtual Observatory (<http://www.us-vo.org>) and the TeraGrid Science gateway (<http://www.teragrid.org>)).

## **2.1.2 Introduction to Cloud Computing**

### **2.1.2.1 Definition of Cloud Computing**

There are many definitions of cloud computing because its concept is still evolving. [VRCL2008] summarises a set of definitions of cloud computing and gives its own definition: *“Clouds are a large pool of easily usable and accessible virtualised resources (such as hardware, development platforms and/or services). These resources can be dynamically re-configured to adjust to a variable load (scale), allowing also for an optimum resource utilisation. This pool of resources is typically exploited by a pay-*

*per-use model in which guarantees are offered by the Infrastructure Provider by means of customised service level agreements”.*

### **2.1.2.2 Features of Cloud Computing**

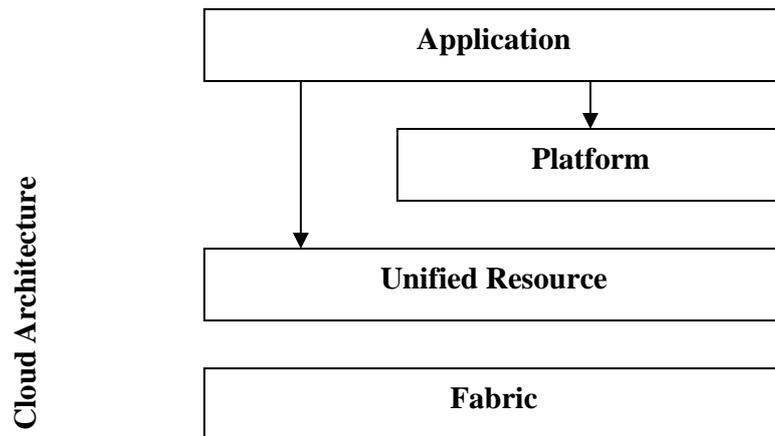
Cloud computing has the following features:

- *Lower Cost.* As infrastructure is typically provided by a third-party and does not need to be purchased for one-time or infrequent intensive computing tasks, cloud computing lowers barriers to entry greatly [BMQL2007].
- *Incremental Scalability.* Cloud environments allow users to access additional computing resources on-demand in response to increased application loads [Gee2008].
- *Reliability and Fault-Tolerance.* Cloud environments capitalise on the built-in redundancy of the large number of servers that make them up by enabling high levels of availability and reliability for applications that can take advantage of this [BYV2008].
- *Service-oriented.* The cloud is a natural home for service-oriented applications, which need a way to easily scale as services get incorporated into other applications [BYV2008].
- *Utility-based.* Users only pay for the services they use, either by subscription or transaction-based models [GK2008].
- *Virtualisation.* Large amounts of computing resources can be provisioned and made available for new applications within minutes instead of days or weeks. Developers can gain access to these resources through a portal and put them to use immediately [BMQL2007].
- *SLA-driven.* Clouds are managed dynamically based on service-level agreements that define policies like delivery parameters, costs, and other factors [BYV2008].

- *User-centric interfaces.* Cloud services could be accessed with user-centric interfaces, which means that the cloud interfaces do not force users to change their working habits, the cloud client required to be installed locally is lightweight, and cloud interfaces are location independent and can be accessed by some well established interfaces like Web service and Internet browser [WTKC+2008].
- *On-demand service provision.* Computing clouds provide resources and services for users on-demand. Users can customise required computing environments later on, for example, software installation, network configuration, as users normally own “root” privilege [WTKC+2008].
- *QoS guaranteed offer.* The computing environments provided by computing clouds can guarantee QoS for users, e.g., hardware performance like CPU bandwidth and memory size [WTKC+2008].
- *Autonomous.* The computing cloud is an autonomous system and managed transparently to users. Hardware, software and data inside clouds can be automatically reconfigured, orchestrated and consolidated to a single platform image, finally rendered to users [WTKC+2008].
- *Pay as you go.* The ability can be provided to pay for the use of computing resources on a short-term basis as needed (e.g., processors by the hour and storage by the day) and release them as needed [AFGJ+2009].

### 2.1.2.3 Architecture of Cloud

Ian Foster et al. proposed a four-layer architecture for cloud computing in comparison to the grid architecture (see Figure 2.2) [FZRL2008].



**Figure 2.2 Cloud Architecture [FZRL2008]**

The fabric layer contains the raw hardware level resources, such as computing resources, storage resources, and network resources.

The unified resource layer contains resources that have been abstracted/encapsulated (usually by virtualisation) so that they can be exposed to upper layer and end users as integrated resources, for instance, a virtual computer/cluster, a logical file system, a database system, etc.

The platform layer adds on a collection of specialised tools, middleware and services on top of the unified resources to provide a development and/or deployment platform, for instance, a Web hosting environment, a scheduling service, etc.

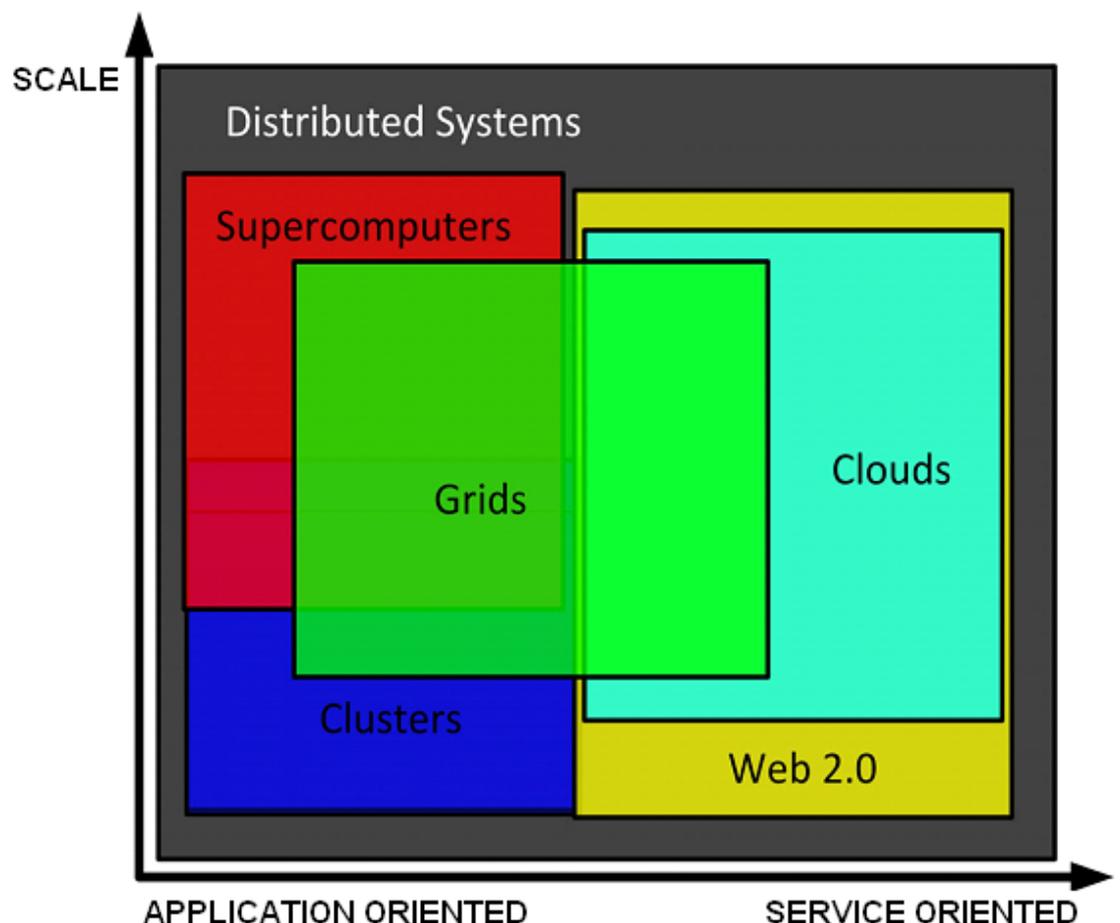
Finally, the application layer contains the applications that would run on the cloud.

### **2.1.3 Relationship of Cloud Computing and Grid Computing**

Foster et al. present an overview of the relationship between the cloud environment and the other domains that it overlaps with as shown in Figure 2.3 [FZRL2008]. As depicted in the figure, Web 2.0 [Ore2007] covers the service-oriented spectrum, where cloud computing lies on the large-scale side. Supercomputing and clusters have been more focused on the traditional application spectrum. From this figure we can see that grid computing indeed overlaps with cloud computing at the service oriented spectrum.

In general, cloud computing is enabled by grid computing, virtualisation, utility computing, hosting and software as a service (SaaS). This fact makes grid computing and cloud computing closely related to each other on one hand, yet different to each other on the other hand.

On one hand, clouds and grids share a lot commonality in their vision, architecture and technology. In fact, it is possible for clouds to be implemented over existing grid technologies leveraging more than a decade of community efforts in standardisation, security, resource management, and virtualisation support [FZRL2008].



**Figure 2.3 Grids and Clouds [FZRL2008]**

On the other hand, clouds and grids differ in various aspects such as security, programming model, business model, compute model, data model, applications, and abstractions [FZRL2008]. Cloud computing can be seen as a natural next step from the

grid-utility model. Modern grid computing technologies have evolved as a way to harness inexpensive servers in a data centre to solve a variety of business problems. Traditionally, grids have lacked the automation, agility and simplicity characterised by cloud computing.

## **2.2 Introduction to Workflow and Workflow Management Systems**

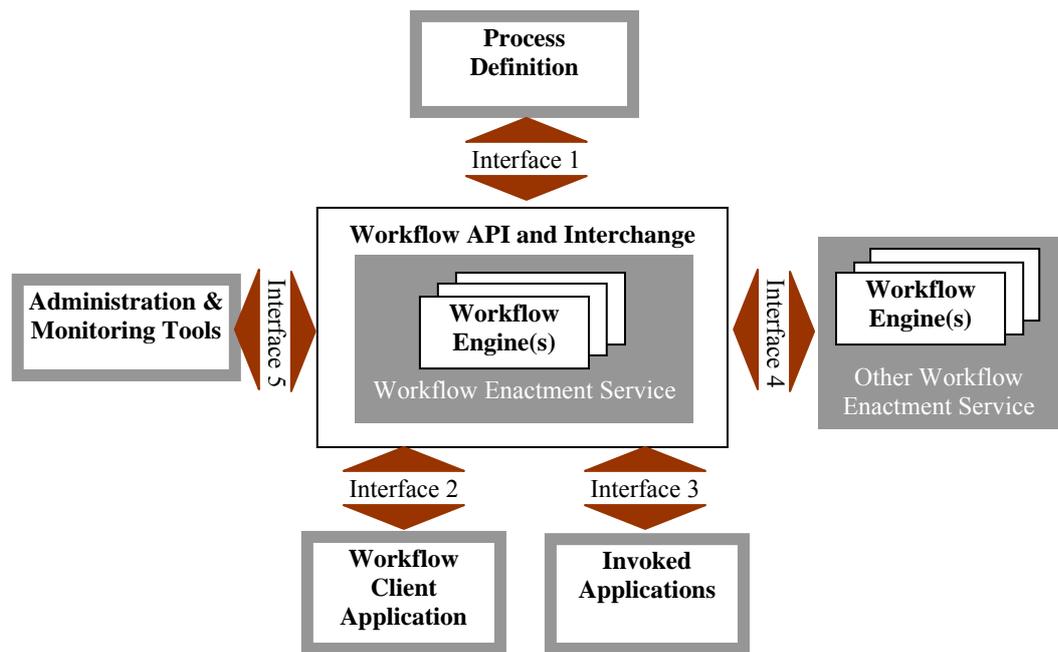
### **2.2.1 Definition of Workflow**

The WfMC (Workflow Management Coalition) defined a workflow as *“the automation of a business process, in whole or part, during which documents, information or tasks are passed from one participant to another for action, according to a set of procedural rules.”* [All2001]

Workflow management is a fast evolving technology which is increasingly being exploited by businesses in a variety of industries. Its primary characteristic is the automation of processes involving combinations of human and machine-based activities, particularly those involving interaction with IT applications and tools. Although its most prevalent use is within the office environment in staff intensive operations such as insurance, banking, legal and general administration, etc, it is also applicable to some classes of industrial and manufacturing applications [DL2008].

### **2.2.2 Architecture of Workflow Management Systems**

WfMC published its reference model in [WfMC95], identifying the interfaces within this structure which enable products to interoperate at a variety of levels. This model defines a workflow management system and the most important system interfaces (see Figure 2.4).



**Figure 2.4 WfMC's Workflow Reference Model**

- *Workflow Engine.* A software service that provides the run-time environment in order to create, manage and execute workflow instances.
- *Process Definition.* The representation of a workflow process in a form which supports automated manipulation.
- *Workflow Interoperability.* Interfaces to support interoperability between different workflow systems.
- *Invoked Applications.* Interfaces to support interaction with a variety of IT applications.
- *Workflow Client Applications.* Interfaces to support interaction with the user interface.
- *Administration and Monitoring.* Interfaces to provide system monitoring and metric functions to facilitate the management of composite workflow application environments.

It can be seen that scheduling is a function module of the Workflow Engine(s), thus it is a significant part of workflow management systems.

## **2.2.3 Existing Workflow Management Systems**

### **2.2.3.1 Conventional Client-Server Workflow Management Systems**

In the earlier stage of the evolution of workflow management, client-server is the most dominant architecture for workflow management systems. This architecture has the advantages such as thin clients, centralised monitoring and auditing, simple synchronisation mechanisms, one copy of process state, and ease of design, implementation and deployment for workflows. However, it also faces conspicuous difficulties like low performance, reliability and scalability. Here are some typical client-server workflow management systems: Exotica/FMQM [AMGA+95], ADEPT [RRD03], DartFlow [CGN96], METUFlow [GACT+97] and IBM ImagePlus [IBM1997]

### **2.2.3.2 Peer-to-peer Workflow Management Systems**

Peer-to-peer workflow management systems abandon the dominating client-server architecture and use the peer-to-peer infrastructure to provide decentralised workflow support. As the workflow functions are fulfilled through the direct communication and coordination among the relevant peers, performance bottlenecks are likely to be eliminated whilst increased resilience to failure and enhanced scalability are likely to be achieved. Here are some typical peer-to-peer workflow management systems: RainMan [PPC97], Serendipity-II [GAHM98] and SwinDeW [YYR2006].

### **2.2.3.3 Grid Workflow Management Systems**

The workflow enactment service of grid workflow management systems may be built on top of the low level grid middleware (e.g. Globus toolkit [Glo2005], UNICORE [AS1999] and Alchemi [LBRV2005]), through which the workflow management system invokes services provided by grid resources [FG2006]. At both the build-time and run-time stages, the information about resources and applications may need to be retrieved using grid information services. There are many grid workflow management systems currently, and here we only list several representative grid workflow systems which are in use recently. These systems include ASKALON [FJPP+2005], GrADS

[BCCD+2001], GridAnt [LAHZ+2004], Gridbus [BV2004], GridFlow [CJSN2003], Karajan [LH2005], Kepler [ABJJ+2004], Pegasus [DBGK+2003], Taverna [OAFM+2004] and Triana [CGHM+2005]. The details of these workflow management systems can be found in their respective references. In [YB2005], comparisons of several representative Grid workflow systems are given in aspects of (1) scheduling architecture, (2) decision making, (3) planning scheme, (4) scheduling strategy, and (5) performance estimation.

#### **2.2.3.4 Cloud Workflow Management Systems**

Due to the relative new appearance of cloud computing, there are very few publications thus far on cloud workflow management systems, though we are aware that many researchers are working in this area at the moment. The following pieces of work involve executing workflows on clouds. In [PCVR+2008], Aneka [BYV2008] is used as a cloud middleware on which the Gridbus workflow management system [YB2004] deploys and manages job execution. In [HMFD+2008], Hoffa et al. explore the differences between running scientific workflows on the cloud and running them on the grid, using Montage [BDGJ+2004] on top of the Pegasus-WMS software [DSSB+2005].

#### **2.2.3.5 Our Own Workflow Management Systems for Grid and Cloud**

To enable effective research in this area, we have developed a family of workflow management systems for testing and simulation. Most of the simulations in this thesis are performed on SwinDeW-G (Swinburne Decentralised Workflow for Grid) [YLCL+2007] and SwinDeW-C (Swinburne Decentralised Workflow for Cloud) [YLCL+2008].

SwinDeW-G is a novel peer-to-peer (P2P) based grid workflow system incorporating P2P and grid technologies for taking advantages of both. It is realised based on the former SwinDeW P2P based workflow system with peers packaged as grid services to reduce the development cost. The utilisation of the grid technology provides more power to handle sophisticated workflow applications while the facilitation of the P2P technology improves the performance and increases the reliability and scalability of the system.

SwinDeW-C is a new cloud workflow system. It uses cloud computing technology to facilitate the execution of workflows. In our design, it is divided into the following major parts: dynamic service clouds, cloud workflow execution agents, a services catalogue, and a user interface.

### **2.3 Introduction to Workflow Scheduling Algorithms**

Workflow scheduling is one of the key issues in the workflow management [YB2005], especially in the grid and cloud workflow systems. It is a process that maps and manages the execution of inter-dependent tasks on the distributed resources. It allocates suitable resources to workflow tasks such that the execution can be completed to satisfy objective functions imposed by users.

Proper scheduling can have significant impact on the performance of the system. However, in general, the problem of mapping tasks on distributed services belongs to a class of problems known as NP-hard problems [Ull1975]. For such problems, no known algorithms are able to generate the optimal solution within polynomial time. Even though the workflow scheduling problem can be solved by using exhaustive search, the complexity of the methods for solving it is very high. In grid environments, scheduling decisions must be made in the shortest time possible, because there are many users competing for resources, and time slots desired by one user could be taken by another user at any moment [YB2007]. This equally applies to cloud computing environments.

To date, there are two major types of workflow scheduling, best-effort based and QoS constraint based scheduling [YB2007], primarily for grid workflow management systems. Best-effort based scheduling attempts to minimise the execution time, ignoring other factors such as the monetary cost of accessing resources and various users' QoS satisfaction levels. In contrast, QoS constraint based scheduling attempts to maximise the performance under QoS constraints, for example time minimisation under budget constraints. We discuss them in the following two sections respectively.

### **2.3.1 Best-effort Based Workflow Scheduling Algorithms**

The representative best-effort based scheduling algorithms are summarised in this section.

#### **2.3.1.1 A Hybrid Heuristic**

In [SZ2004], a hybrid heuristic was proposed for scheduling DAG (Directed Acyclic Graph) on heterogeneous systems. This heuristic combines dependency mode and batch mode in the scheduling procedure. It first computes the rank values of each task and ranks all tasks in descending order of their rank values, and then it creates groups of independent tasks. In the grouping phase, it processes tasks in order of their rank values and add tasks into the current group. Once it finds a task which has a dependency with any task within the group, it creates another new group. As a result, a number of groups of independent tasks are generated. The group number is assigned based on the order of rank values of their tasks, that is, if  $m > n$ , the ranking value of tasks in group  $m$  is higher than that of the tasks in group  $n$ . Then it schedules tasks group by group and uses a batch mode algorithm.

#### **2.3.1.2 TANH**

The TANH (Task duplication-based scheduling Algorithm for Network of Heterogeneous systems) algorithm [BA2004] combines cluster based scheduling and duplication based scheduling. It first traverses the task graph to compute parameters of each node including earliest start and completion time, latest start and completion time, critical immediate parent task, best resource and the level of the task. After that it clusters tasks based on these parameters. The tasks in the same cluster are supposed to be scheduled on the same resource. If the number of the cluster is greater than the number of resources, it scales down the number of clusters to the number of resources by merging some clusters. Otherwise, it utilises the idle times of resources to duplicate tasks and rearrange tasks in order to decrease the overall execution time.

### 2.3.1.3 Dynamic Critical Path for Grids (DCP-G)

In paper [RVB2007], the Dynamic Critical Path for Grids (DCP-G) algorithm is proposed for Scheduling Scientific Workflow Applications on Global Grids.

For a task graph, the lower and upper bounds of starting time for a task are denoted as the *Absolute Earliest Start Time (AEST)* and the *Absolute Latest Start Time (ALST)* respectively. In DCP-G, a task is considered to be on the critical path (a critical task) if its *AEST* and *ALST* values are equal. The critical task selected for scheduling is the one that is on the critical path and has no unmapped parent tasks. After identifying a critical task, the algorithm selects the resource that provides the minimum execution time for that task. This is discovered by checking all the available resources for one that minimises the potential start time of the critical child task on the same resource, where the critical child task is the one with the least difference of *AEST* and *ALST* among all the child tasks of the critical task. Finally, the critical task is mapped to the resource that provides earliest combined start time.

### 2.3.1.4 Fast Critical Path (FCP) Algorithm

The Fast Critical Path (FCP) algorithm [RG1999] intends to reduce the complexity of the scheduling task, while maintaining scheduling performance. This complexity can be effectively reduced by maintaining only a constant size sorted list of ready tasks. The others are stored in an unsorted FIFO list which has an  $O(1)$  time access. When a task becomes ready it is added to the sorted list when there is room to accommodate it, otherwise it is added to the FIFO list. The tasks are always dequeued from the sorted list. After a dequeue operation, if the FIFO list is not empty, one task is moved to the sorted list. The time complexity of sorting tasks using a list of size  $H$  decreases to  $O(H \log H)$  as all the tasks are enqueued and dequeued in the sorted list only once.

However, a possible drawback of using a fixed size for a sorted task list is that the task with the highest priority may not be included in the sorted list, but is temporarily stored in the FIFO list. Thus the size of the sorted list must be large enough not to affect the performance of the algorithm too much. At the same time, it should not be too large in view of the time complexity. The authors' experiments show that a priority list size of  $P$  (number of processors) is a good choice for the FCP algorithm. A smaller size

penalises problems with limited parallelism, while a greater size does not yield further improvements.

### **2.3.1.5 Adaptive Generalised Scheduler (AGS)**

In the Adaptive Generalised Scheduler (AGS) algorithm [AK2005], one of the important classes is the reservation file, which is implemented as a *gapTime* vector. The *gapTime* vector is, therefore, initialised to full availability, in the case all the computing-nodes are available for scheduling the jobs. If some of the nodes have been reserved for certain periods of time, either by the owner of the resource or by other jobs, which had been allocated to the computing-nodes, in an earlier cycle of scheduling, the *gapTime* vector would contain the information.

First, for every task, a start time is defined. The start time is determined by the end time of the last parent process. As a first step, the computing-node on which the last parent was processed is identified. The delays in transferring the data from the other parents to the node are calculated and the start time is updated. If the computing-node is available from the updated start time of the task up to the end of the processing time of the task, the task is allocated to the node. Otherwise for every node the start time is recalculated after evaluating the delays from all the parent processes. Using the updated start time, the end time of the process for the node is found. Out of all the available nodes, the task is mapped to a computing-node, for which the end time is the earliest. After the mapping, the *gapTime* vector is updated.

### **2.3.1.6 Workflow Mapping Mechanism (WMM)**

The main goal of the Workflow Mapping Mechanism (WMM) [KTML+2008] is to find an optimum selection with regard to the QoS metrics requested by the user and offered by the service providers. The major steps of the algorithm are described briefly as follows:

- (1) Calculate auxiliary values (pilots for the algorithm completion) in order to define metrics that “characterise” the level of QoS provided by the service instances.

(2) Initialise workflow mapping with service instances that meet the user's requirement for availability without violating the cost constraint.

(3) Define a service instance (candidate) for each service type. The reason for this step is to discover the candidates that provide higher level of QoS for each service type within a workflow.

(4) Create a list with the "best candidates" for each service type in order to find possible replacement(s).

(5) Schedule a scheme that allows more than one service instance to be selected per service type and is applied to meet the time constraint (deadline) set by the user.

#### **2.3.1.7 Adaptive Workflow Splitting (AWS) Algorithm**

The Adaptive Workflow Splitting (AWS) algorithm [DA2007] is described as follows. When a scheduler receives a job, it first traverses the job's DAG  $G$  to compute its  $CCR$  and the number of resource clusters (denoted as  $N$ ), where  $CCR$  (*communication-to-computation ratio*) is the average ratio of computation cost to communication cost. A high  $CCR$  value means a task graph is computation-intensive. Then, the scheduler selects  $N$  resource clusters that have the highest  $N$  ranks according to its knowledge. A graph partition iteration checks every remaining nodes in  $G$  to determine whether the node can be put into a sub-graph  $G'$ . It starts from looking for the largest communication edge whose two nodes have not been checked yet. If there is no such edge, this means that all unchecked nodes are now isolated from  $G'$  and each other, and these tasks will be merged into the sub-graph if possible. Otherwise, if this edge can be merged into  $G'$ , depth-first graph traversal from the two vertices of this edge will start to merge more task nodes. The objective of the graph traversal is to merge as many tasks as possible into the current sub-graph without violating the constraint of the resource threshold, and at the same time, following precedence orders among tasks to avoid unnecessary break-up which can not increase parallelism, but may increase inter-cluster communication cost.

A stack is used to manipulate task nodes in the traversal. If this edge cannot be merged in  $G'$  under the constraint, the responsible node will be marked out. This means

a new sub-graph is created. The scheduler repeats the loop until all nodes in  $G$  are assigned to resource clusters. Once the algorithm finishes, the scheduler will dispatch every sub-graph to its designated resource cluster.

#### **2.3.1.8 Adaptive Scheduling Algorithm (ASA)**

The Adaptive Scheduling Algorithm (ASA) [AF2008] is proposed to find a suitable execution sequence for workflow activities by additionally considering resource allocation constraints and dynamic topology changes. This algorithm utilises a multi-stage distribution algorithm to cope with network dynamics.

The idea of ASA is to divide the scheduling process into 2 phases: (1) a logical partition step and (2) a physical allocation step. The key motivation behind this multi-stage procedure is to reduce the complexity of scheduling problem. The logical partition step identifies partitions of workflow activities based on similar attribute values and the physical allocation step assigns these partitions by using constraint programming. Meanwhile, network changes are considered in ASA to make it a robust and adaptive scheduling procedure.

#### **2.3.1.9 Heterogeneous Earliest-Finish-Time Algorithm (HEFT)**

The HEFT algorithm [WPF2005] consists of 3 phases: (1) the weighting phase: assigns the weights to the nodes and edges in the workflow; (2) the ranking phase: creates a sorted list of tasks, organised in the order how they should be executed and (3) the mapping phase: assigns the tasks to the resources.

In the weighting phase, weights are assigned to nodes and edges. The weights assigned to the nodes are calculated based on the predicted execution times of the tasks, and the weights assigned to the edges are calculated based on predicted times of the data transferred between the resources.

The ranking phase is performed traversing the workflow DAG upwards, and assigning a rank value to each of the tasks. Rank value is equal to the weight of the node plus the execution time of the successors. The successor execution time is estimated, for

every edge being immediate successors of the node, adding its weight to the rank value of the successive node, and choosing the maximum of the summations.

In the mapping phase, consecutive tasks from the ranking list are mapped to the resources. For each task, the resource which provides the earliest expected time to finish execution is chosen.

#### **2.3.1.10 Greedy Randomised Adaptive Search Procedure (GRASP)**

GRASP (greedy randomised adaptive search procedure) [PR2002] is usually implemented as an iterative procedure, where each iteration is made up of a construction phase and a local search phase.

In the construction phase, the restricted candidate list (RCL) of each unmapped ready task is created, where a ready task is a task whose parent tasks have all been scheduled. The RCL is used to record the best candidates for executing the task, but not necessarily the top candidate of the resources for processing each task. Usually the RCL consists of the  $k$  best rated resources or those whose are rated a value better than a given threshold. Then in the local search phase, the neighbourhood of the current solution is searched to generate a new solution. The new solution will replace the current constructed solution if its overall performance is better (i.e. its makespan is shorter than that of the solution generated) in the construction phase. The iteration continues until certain criteria are met.

#### **2.3.1.11 Simulated Annealing (SA) Algorithm**

The Simulated Annealing (YMND2003) algorithm assumes that an even better solution is more probably derived from good solutions. The input of the algorithm is an initial solution which is constructed by assigning a resource to each task at random, the SA algorithm runs through a number of iterations to find a better solution. During each cycle, it generates a new solution by applying random change to the current solution. The new solution and the current solution are compared. If the new solution has greater benefit value than the current solution it is accepted as the new selection. However, if the new solution has a lower benefit, it is accepted with a probability  $e^{-d\beta/T}$ , where  $d\beta$  is the difference in benefit value between the two solutions, and  $T$  is a control parameter.

This process is repeated with each iteration consisting of either a maximum number of new solutions being accepted, or a maximum number of solutions being considered. At the end of each iteration,  $T$  is decreased. Once an iteration is completed with no new solutions being accepted, the current solution is returned as the best available solution. Low values of  $T$  decrease the probability that a solution with a lower benefit value will be selected, so do large values of  $d\beta$ . At high values of  $T$ , worse solutions are often accepted, reducing the chance of the algorithm getting caught in local maxima. As  $T$  is decreased, so is the regularity with which worse schedules are accepted, allowing the algorithm to settle with the best algorithm found.

#### **2.3.1.12 Genetic Algorithms (GA)**

Genetic algorithms [WSRM1997][SCJH+2004] combine exploitation of the best solutions from past searches with the exploration of new regions of the solution space. Any solution in the search space of the problem is represented by an individual chromosome. A genetic algorithm maintains a population of chromosomes that evolves over generations. The quality of a chromosome in the population is determined by a fitness function, which is the makespan that results from the matching of tasks to machines within that chromosome. The fitness value indicates how good the individual is compared to others in the population.

A typical genetic algorithm is described as follows. First it creates an initial population consisting of randomly generated solutions from a uniform random distribution. After applying genetic operators, namely selection, crossover and mutation, one after the other, new offspring are generated. Then the evaluation of the fitness of each individual in the population is conducted. The fittest individuals are selected to be carried over to the next generation. The above steps are repeated until the termination condition is satisfied. Typically, a genetic algorithm terminates after a certain number of iterations, or if a certain level of fitness value has been reached.

#### **2.3.1.13 Myopic Algorithm**

The Myopic algorithm [WPF2005] is the simplest scheduling method for scheduling workflow applications and it makes schedule decision based only on one individual task.

It tries to schedule a ready task whose parent tasks have all been scheduled to a resource which is expected to complete the task earliest. This procedure repeats until all tasks have been scheduled.

#### 2.3.1.14 Min-Min, Max-Min and Sufferage Heuristic

Min-Min heuristic [MASH+1999] makes decisions based on a set of parallel independent tasks. For each iterative step, it calculates the earliest completion time (ECT) of each task on its every available resource and obtains the minimum ECT (MCT) for each task. The task with the minimum MCT value over all tasks is chosen to be scheduled first at this iteration.

More specifically, let  $r_j$  denote the *expected time*, machine  $m_j$  will become ready to execute a task after finishing the execution of all tasks assigned to it at that point of time. First the *expected completion time* for task  $t_i$  on machine  $m_j$  (denoted as  $c_{ij}$ ) are computed using the *expected execution time* for task  $t_i$  on machine  $m_j$  (denoted as  $e_{ij}$ ) and  $r_j$  values. For each task  $t_i$  the machine that gives the earliest *expected completion time* is determined by scanning the rows of matrix  $c$  which composes of all  $e_{ij}$ . Task  $t_k$  that has the minimum earliest expected completion time is determined and then assigned to the corresponding machine. Matrix  $c$  and vector  $r$  which composes of all  $r_j$  are updated and the above process is repeated with tasks that have not yet been assigned a machine.

Max-Min heuristic is very similar to the Min-Min heuristic. The only difference is the Max-Min heuristic will schedule the task that has the maximum earliest expected completion time, rather than that which has the minimum earliest expected completion time in Min-Min heuristic, on the resource which is expected to complete the task at earliest time.

Sufferage heuristic is based on the idea that better mappings can be generated by assigning a machine to a task that would “suffer” most in terms of expected completion time if that particular machine is not assigned to it. Let the *sufferage value* of a task  $t_i$  be the difference between its second earliest completion time (on some machine  $m_y$ ) and its earliest completion time (on some machine  $m_x$ ). That is, using  $m_x$  will result in the best completion time for  $t_i$ , and using  $m_y$  the second best.

More specifically, Sufferage heuristic first finds the machine  $m_j$  that gives the *earliest completion time* for task  $t_k$ , and tentatively assign  $m_j$  to  $t_k$  if  $m_j$  is unassigned. Then it marks  $m_j$  as assigned, and removes  $t_k$  from meta-task. If, however, machine  $m_j$  has been previously assigned to a task  $t_i$ , choose from  $t_i$  and  $t_k$  the task that has the higher *sufferage value*, assign  $m_j$  to the chosen task, and remove the chosen task from the unscheduled task list. The unchosen task will not be considered again for this scheduling round, but shall be considered for next scheduling round.

## **2.3.2 QoS-constraint Based Workflow Scheduling Algorithms**

The representative QoS constraint based scheduling algorithms are summarised in this section. As we can find that since the most popular QoS constraint is cost constraint, most QoS constraint based scheduling algorithms focus on cost constraint correspondingly.

### **2.3.2.1 Back-tracking Algorithm**

The back-tracking algorithm [MC2004] assigns available tasks to the least expensive computing resources. An available task is an unmapped/unscheduled task whose parent tasks have been scheduled. If there is more than one available task, the algorithm assigns the task with the largest computational demand to the fastest resources in its available resource list. The heuristic repeats the procedure until all tasks have been mapped. After each iterative step, the execution time of the current assignment is computed. If the execution time exceeds the time constraint, the heuristic back tracks the previous step, removes the least expensive resource from its resource list and reassigns tasks with the reduced resource set. If the resource list is empty the heuristic keeps back tracking to the previous steps, reduces corresponding resource list and reassigns the tasks.

### **2.3.2.2 Loss and Gain Approach**

The LOSS and GAIN approach [SZTD2005] iteratively adjusts a schedule which is generated by a time optimised heuristic or a cost optimised heuristic to meet the budget constraints. It starts with an initial assignment of the tasks onto machines (schedule) and

computes for each reassignment of each task to a different machine based on a weight value associated with that particular change. Those weight values are tabulated; thus, a weight table is created for each task in the DAG (Directed Acyclic Graph) and each machine. Two alternative approaches for computing the weight values are proposed, depending on the two choices used for the initial assignment: either optimal for makespan or cost. Using the weight table, tasks are repeatedly considered for possible reassignment to a machine, as long as the cost of the current schedule exceeds the budget (in the case that loss is followed), or, until all possible reassignments would exceed the budget (in the case of gain). In either case, the algorithm tries to reassign any given pair of tasks only once, so when no reassignment is possible the algorithm will terminate.

### **2.3.2.3 Genetic Algorithm Approach**

A genetic algorithm based approach [YB2006] is developed to solve the deadline and budget constrained scientific workflow scheduling problem. As the goal of the scheduling is to maximise the performance based on two factors: time and monetary cost, the fitness function of the genetic algorithm [WSRM1997] separates into two parts: *cost-fitness* and *time-fitness*. Both functions use two binary variables,  $\alpha$  and  $\beta$ . If users specify a budget constraint, then  $\alpha=1$  and  $\beta=0$ . If users specify a deadline, then  $\alpha=0$  and  $\beta=1$ . For budget constrained scheduling, the cost-fitness component encourages the formation of the solutions that satisfies the budget constraint. For deadline constrained scheduling, it uses the genetic algorithm to choose schedules with less cost.

This genetic algorithm approach either minimises the monetary cost while meeting users' budget constraint, or minimises the execution time while meeting users' deadline constraints. Compared with most other existing genetic algorithms, the proposed approach targets heterogeneous and reservation based service oriented environments for solving budget and deadline constrained optimisation problems.

### **2.3.2.4 Improved Genetic Algorithm**

Improved Genetic Algorithm (Improved GA) is proposed in [XCY2007] to optimise the allocation of workflow resources with cost constraint to minimise the average throughput time of a workflow. The improved GA follows the following steps:

- (1) Initialise parameters: population size:  $np$ , max generation:  $ng$ , crossover rate:  $pc$ , mutation rate:  $pm$ ;
- (2) Create a random initial population  $S$ ;
- (3) Obtain the greedy solution;
- (4) Stop if the number of generations reaches the value of max generation  $ng$ ; otherwise go ahead;
- (5) Evaluate each member of the current population  $S$  by calculating its fitness;
- (6) Apply selection;
- (7) Apply crossover and improving;
- (8) Apply mutation, mending and improving;
- (9) Return to Step 4.

#### **2.3.2.5 Ant Colony System (ACS) Algorithm**

An Ant Colony System (ACS) algorithm [CZ2009] can be used for workflow applications in market-driven or economy-driven grids under the service-oriented architecture. In the algorithm, different QoS parameters are considered, including reliability, time, and cost. Users are allowed to define QoS constraints to guarantee the quality of the schedule. Moreover, the optimising objective of the algorithm is based on the user-defined QoS preferences. The ACS algorithm can be viewed as the interplay of the following procedures:

- (1) Initialisation of algorithm: All pheromone values and parameters are initialised at the beginning of the algorithm.

(2) Initialisation of ants: A group of  $M$  artificial ants are used in the algorithm. In each iteration, each ant randomly selects a constructive direction and builds a sequence of tasks.

(3) Solution construction:  $M$  ants set out to build  $M$  solutions to the problem based on pheromone and heuristic values using the selection rule of the ACS algorithm.

(4) Local pheromone updating: Soon after an ant maps a service instance to task, the corresponding pheromone value is updated by a local pheromone updating rule.

(5) Global pheromone updating: After all ants have completed their solutions at the end of each iteration, pheromone values corresponding to the best-so-far solution are updated by a global pheromone updating rule.

(6) Terminal test: If the test is passed, the algorithm will be ended. Otherwise, go to step 2 to begin a new iteration.

#### **2.3.2.6 Deadline Distribution Algorithm**

The deadline distribution algorithm [YBT2005] tries to minimise the cost of execution while meeting the deadline for delivering results. This algorithm partitions a workflow and distributes the overall deadline into each task based on their workload and dependencies. It uses Synchronisation Task Scheduling (STS) for synchronisation tasks and Branch Task Scheduling (BTS) for branch partition respectively. Once each task has its own sub-deadline, a local optimal schedule can be generated for each task. If each local schedule guarantees that their task execution can be completed within their sub-deadlines, the whole workflow execution will be completed within the overall deadline. Similarly, the result of the cost minimisation solution for each task leads to an optimised cost solution for the entire workflow. Therefore, an optimised workflow schedule can be constructed from all local optimal schedules. The schedule allocates every workflow task to a selected service such that it can meet its assigned sub-deadline at a low execution cost.

## **2.4 Problem and Requirements Analysis**

### **2.4.1 Example Scenario of Bank Cheque Processing**

In a bank cheque processing scenario, there are millions of concurrent cheque-processing transactions per day, while each of them is a rather simple workflow instance with only a few steps. This cheque-processing procedure can be modelled as an instance-intensive workflow. If cheque-processing needs image transmission for authentication, it can be further modelled as instance-intensive workflows involving significant communication overhead. Moreover, if the cheque-processing procedure is executed on rented services within a given budget, they can be modelled as cost-constrained instance-intensive workflows.

### **2.4.2 Problem Analysis**

However, instance-intensive (business/scientific) workflows are not sufficiently researched particularly in comparison to computation-intensive scientific workflows. Most existing workflow scheduling algorithms are not specifically for instance-intensive workflows, thus it is necessary to develop dedicated workflow scheduling algorithms which take the characteristics of instance-intensive workflows into account.

As the best-effort based workflow scheduling algorithms like Hybrid Heuristic, TANH, DCP-G, FCP, AGS, WMM, AWS, ASA and HEFT are designed for scheduling of single complex DAG, they are not suitable for scheduling instance-intensive workflows where the scheduling is for a huge number of concurrent but relatively simple instances. As we take a look at algorithms like the GRASP algorithm, Simulated Annealing (SA) algorithm and genetic algorithms (GA), all of them obtain a global solution by comparing differences between randomised schedules over a number of iterations. Considering the large number of concurrent workflow instances in instance-intensive workflows, the scheduling time spent on iterations for this huge number of workflow instances is clearly very high. As for other algorithms, the decision of the Myopic algorithm is based on one task and hence not suitable for scheduling instance-intensive workflows which focuses on the overall throughput in overall terms. Compared with the algorithms above, the simple Min-Min algorithm is more applicable for scheduling instance-intensive grid workflows due to their batch processing mode.

However, the strategy needs to be adjusted to adapt to the characteristics of instance-intensive cloud workflows, with or without involving considerable communication overheads.

As for the QoS constraint based workflow scheduling algorithms, the back-tracking algorithm does not consider resource competition with other activities, including the activities in other concurrent instances as in instance-intensive workflows, or other non-workflow activities as in a shared cloud computing environment. The LOSS and GAIN approach, the genetic algorithm approach, improved GA and ACS spend significant time on their iteration steps, thus are not suitable for scheduling instance-intensive workflows which have a huge number of concurrent instances. The deadline distribution algorithm is designed for single instance on utility grid rather than the instance-intensive scenario on the cloud computing platform. However, the philosophy of this algorithm can be borrowed and extended for scheduling instance-intensive cost-constrained cloud workflows.

### **2.4.3 Requirements Analysis**

Considering the characteristics of instance-intensive workflows and cloud computing environments, the new scheduling algorithms in our research should meet the following requirements:

- The overall performance of the system is more important for instance-intensive cloud workflow scheduling. As there is a huge number of concurrent, but relatively simple, workflow instances that need to be scheduled, increasing the overall throughput is the most important factor, rather than decreasing the execution time of a single instance as most other workflow scheduling algorithms pursue.
- Different infrastructures of the cloud computing environments should be treated differently when designing scheduling algorithms for cloud workflows. (1) In the case of a cloud computing environment composed of physical collocated grids via the Internet, the communication cost within each grid is usually negligible. In such circumstance, how to balance the load among grids and

increase the utilisation rates inside each grid becomes an important requirement.

(2) In the case of a service-based cloud computing environment composed of services scattered on the Internet, the communication overheads can no longer be ignored. In such circumstance, dynamic adaptation to the network bandwidth for communication should be reflected in the workflow scheduling algorithms.

(3) In the case of commercial cloud computing environment, the essence of “pay per use” makes the execution cost another important factor which usually needs to compromise with the execution time. Thus if the execution cost needs to be considered, how to compromise it with the execution time is an important factor.

## **2.5 Summary**

In this chapter, the background of related technologies of grid and cloud computing environments, workflow management systems and workflow scheduling algorithms is presented. The problem and requirements of this research have been analysed. Although there exist many workflow scheduling algorithms, none of them are designed for instance-intensive cloud workflows. The three major reasons are summarised as follows. First, many of them are designed for scheduling a single workflow instance for minimising the execution time, however, the overall throughput is the most important factor in overall terms for instance-intensive workflows. Second, the characteristics of instance-intensive cloud workflows usually are not considered, such as the resource competitions with workflow activities of other instances or other non-workflow activities. Third, scheduling itself in some algorithms, especially those genetic based algorithms, involves time-consuming iterations which are definitely not suitable for instance-intensive cloud workflows. For these reasons, we should develop a set of new scheduling algorithms for these workflows.

## **Chapter 3**

# **Throughput Maximisation Strategy for Scheduling Instance-intensive Cloud Workflows**

This chapter is based on our paper [LCYJ2008] and is organised as follows. In Section 3.1, a brief introduction to the background of the proposed strategy is given, followed by the requirements analysis of scheduling for instance-intensive workflows in a grid-based cloud computing environment in Section 3.2. Section 3.3 presents the scheduling infrastructure of the proposed strategy: Throughput Maximisation Strategy (TMS), and Section 3.4 discusses the details of the TMS strategy. In particular, Section 3.4.1 and Section 3.4.2 present the OAL (Opposite Average Load) algorithm and the EMM (Extended Min-Min) algorithm, the two algorithms that comprise TMS. Then Section 3.5 presents several examples to illustrate the advantages of the TMS strategy. Later Section 3.6 demonstrates the benefits of our work via simulation and comparison, and finally in Section 3.7, we summarise this chapter.

### **3.1 Introduction**

The wide use of the Internet and the promotion of cloud computing are changing our ways of thinking. The prosperity of e-business requires the ability to process instance-intensive workflows. First of all, the most notable characteristic of these workflows is the large amount of concurrent relatively simple workflow instances and the fierce competition of resources. Therefore, the new algorithms should not only consider the completion time of each single instance, but most importantly, the overall performance.

Second, the infrastructure of cloud computing platform should also be considered. The most common infrastructures of cloud computing environment consists of physically collocated grids (grid-based) and cloud computing environment consists of services scattered on the Internet (service-based). In this chapter we focus on scheduling instance-intensive workflows on grid-based cloud computing platform, scheduling on service-based cloud computing platform will be discussed in Chapter 4, and scheduling on business-based cloud computing platform will be discussed in Chapter 5.

## **3.2 Requirements Analysis and Overall Solution**

### **3.2.1 Requirements Analysis**

As mentioned in 2.4.2, most existing workflow scheduling algorithms are designed for the scheduling of a single complicated workflow instance rather than for multiple instance-intensive workflows. Therefore, we need to develop new solutions for the scheduling of instance-intensive e-business workflows on cloud computing environment consists of physically collocated grids.

The new algorithms must be able to support the scheduling of multiple instances of multiple workflows, because there are usually millions of instances running concurrently in an e-business environment. How to complete as many workflow instances in as short time as possible becomes the most important issue. Thus we need to design a new best-effort scheduling algorithm for such workflows.

The overall performance can be defined in many aspects. Among them, we focus on the following aspects according to the characteristics of instance-intensive workflows:

- *Overall load balance*: it is obvious that an overall load balanced system can provide better QoS (Quality of Service) for most users. Thus one goal of the proposed strategy is to achieve an overall load balance.
- *Resource utilisation*: the algorithms need to manage the resources efficiently to deal with the fierce competitions of resources of the huge number of workflow instances.

So increasing the utilisation rate of resources is another important issue of the proposed strategy.

- *Overall throughput*: the ultimate goal of instance-intensive workflow scheduling is to maximise the overall throughput, even with a slight QoS degradation for some single instances.

### **3.2.2 Overall Solution**

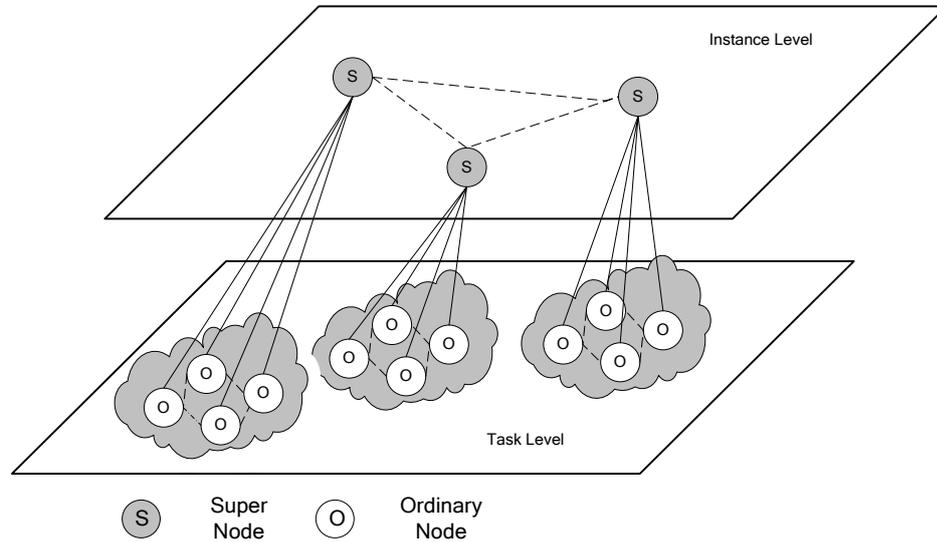
To achieve the goals mentioned before, we propose a Throughput Maximisation Strategy (TMS) which contains two specific scheduling algorithms in different layers for instance-intensive workflows. In detail, the OAL (Opposite Average Load) algorithm mainly deals with the instance scheduling and tries to maximise the overall throughput by pursuing overall load balance. The EMM (Extended Min-Min) algorithm primarily deals with the task scheduling and tries to further maximise the throughput by increasing the utilisation rate of resources within each local autonomous group. By increasing the load balance and resource utilisation rate, the increase of overall throughput is guaranteed.

## **3.3 Scheduling Infrastructure**

In this section, we demonstrate the scheduling infrastructure of TMS before we detail the two algorithms which consist of the strategy in Section 3.4.

As Figure 3.1 shows, the scheduling system is divided into two layers. We can see the scheduling clouds are located at the lower layer. Each scheduling group consists of a super node and some ordinary nodes. The super node is in charge of the management of the scheduling cloud and tasks scheduling, while the ordinary nodes are responsible for the actual execution of tasks. At the same time, the super nodes form the upper layer with each other. The groups at the lower layer are usually formed according to physical location. For example, the nodes in a physically collocated grid are more likely to form a scheduling cloud. It should be noted that although the nodes in a scheduling group are managed by the super node in a centralised fashion for the scheduling purpose, the

communication among nodes is kept in a P2P manner to take advantage of the P2P technology.



**Figure 3.1 Scheduling Infrastructure of SwinDeW-Cg**

As we can see from the top of the figure, all super nodes are organised into a mesh on the upper layer. On this level the workflow instances are scheduled by the cooperation of super nodes to achieve overall load balance. The relationship among super nodes is loosely coupled due to the fact that grid is usually dynamic with nodes joining or leaving at any time. Each super node maintains a neighbour list and exchanges necessary information with their neighbours periodically for maintenance and scheduling purposes.

Considering this system design and the characteristics of instance-intensive workflows, we divide the scheduling process into two stages accordingly. On the first stage, considering the large amount of concurrent workflow instances, it would be better to distribute them evenly before scheduling them directly in the groups which they are submitted to. In fact, before an instance is actually scheduled inside a group, the super node will decide whether to accept it or pass it to another super node according to applied algorithms. Generally speaking, the purpose of these algorithms is to achieve overall load balance. On the second stage, the accepted instance will be submitted to the selected group. Every group is an autonomous scheduling region with the super node controlling all the resources belonged to it. The centralised control grants the super node

the ability to manage the resources efficiently so that it can deal with the fierce competitions for resources. Of course, the ultimate goal of scheduling at this level is to further maximise the overall throughput, even with a slight QoS degradation for some single instances.

### **3.4 Opposite Average Load (OAL) and Extended Min-min (EMM) Algorithms**

In this section, we detail the two scheduling algorithms, namely, the OAL (Opposite Average Load) and EMM (Extended Min-Min) scheduling algorithms. The former is designed to achieve overall load balance and the latter mainly maximises the utilisation rate of resources. As mentioned before, these two algorithms are realised on the upper layer and the lower layer respectively. We discuss them in Sections 3.4.1 and 3.4.2 respectively.

#### **3.4.1 OAL (Opposite Average Load) Scheduling Algorithm**

At instance level, each super node uses a specific scheduling algorithm to achieve the overall load balance at the instance level, for example, the gossip based scheduling algorithm. This algorithm is the most popular one used by many practical distributed systems, and will be denoted as the original Gossip algorithm in the remainder of this chapter.

A typical Gossip algorithm proceeds at two phases. For workflow scheduling, in the first phase, each super node chooses some random neighbours and exchanges load values with them in parallel. Then, at the second phase, each super node inserts its load value into the messages it has received and then propagates them in the next round. In this way every super node knows the load value of its neighbours. Thus when an instance is submitted to the super node, it will accept the workflow instance if it is the least loaded super node of its surrounding area; or otherwise pass the instance to its least loaded neighbour.

On one hand, the original Gossip algorithm is designed for ordinary workflows scheduling and schedule each instance one by one. However, considering the large number of workflow instances in instance-intensive workflows, we must change the one

by one mode to a batch mode. On the other hand, we can use different parameters for a better scheduling result. Thus we can improve the algorithm above as follows.

In our algorithm, each super node maintains the load value of the group it manages and the OAL (Opposite Average Load) value of its direct neighbours. For a specific node  $A$ , the OAL value of its neighbour  $B$  is the average load value of  $B$  and all its neighbours except  $A$ . The OAL can present the average load of the opposite direction which can guide the scheduling procedure later.

To prevent instances from being distributed back, we assign a distance value from the source to each corresponding node. In fact, the distance measure procedure should be done first. At the beginning, the source will send a distance message which contains the source and the distance of the sender from the source to each direct neighbour. For the source node, the initial distance is set to 0. Every node who receives this message will check if it has ever received such message before. If not, it will simply set the distance of this neighbour from the source to the value which is contained in the message and its own distance from the source to this value plus 1. Otherwise, if it has already been received before and the new value is smaller, it will update the distance of this neighbour from the source to the smaller value. If this new value plus 1 is smaller than its distance from the source, it will update this value correspondingly.

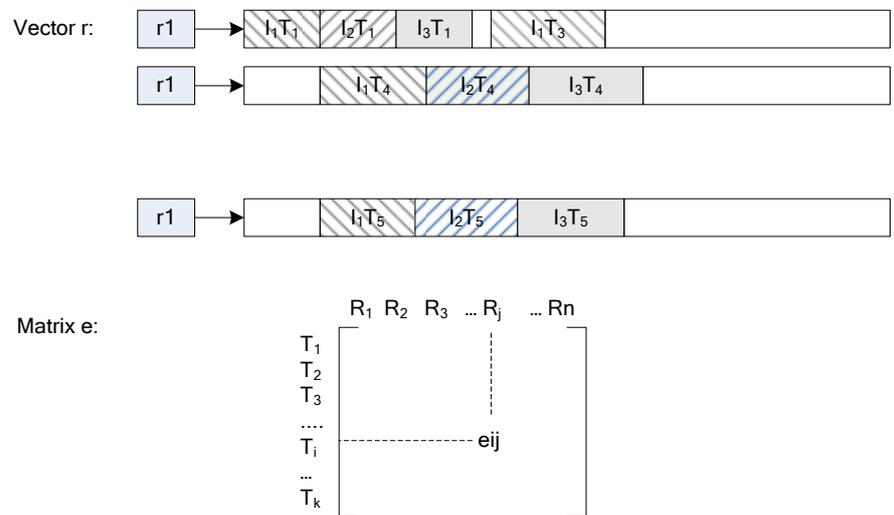
To complete the distance measure process in time, we limit the maximum distance that a message can be relayed. When a node receives such messages, if it has to update its distance from the source (either receiving for the first time or receiving a shorter distance) and the distance contained in the message does not exceed the predefined maximum value, it will send new distance messages to all its neighbours telling them this information. Otherwise it will simply ignore such messages. It should be addressed that the distance measure procedure will only have been executed once for a node when it is needed and the result can be cached for a certain time for future use.

When the distance measure has been completed, every nearby node will know the distance of itself and its neighbours from the source, thus the scheduling procedure can start. Suppose a number of workflow instances have been submitted to the super node, it will calculate the proportion which will be accepted and the proportion which will be

distributed to other neighbours based on its own load value and the OAL value of its neighbours. The algorithm only considers distribution to itself and those neighbours whose distances from the source are greater than its own distance. The distribution procedure is similar to the Greedy algorithm and always seeks the node with the smallest value (load value for itself and OAL value for neighbours), then assigns an instance to this node, and modifies the load value until all tasks are distributed. Finally the super node will send the result to those neighbours who have instances being distributed on.

For each node, if a number of instances have been distributed on itself, it will send a request message to the source node to inform how many instances it wishes to receive. Only after the source node received this message, will the real instance transmission begin.

### 3.4.2 EMM (Extended Min-Min) Scheduling Algorithm



**Figure 3.2 Data Structure Used in EMM algorithm**

At the bottom level, we design an algorithm for task scheduling with the main purpose to achieve maximum throughput in the group. This algorithm is executed periodically to provide dynamic scheduling in a batch mode. We called it the EMM (extended Min-Min) algorithm. The EMM algorithm extends the original Min-Min algorithm and is more suitable for instance-intensive workflows.

The reason for choosing the Min-Min algorithm as the base algorithm is due to an experimental fact. In [BSB2001], eleven static heuristics were compared. For different situations, implementations, and parameter values, Genetic Algorithms consistently gave the best results, and the much simpler Min-Min algorithm usually gave the second best results, with the average performance always within 12% of that of the Genetic Algorithms [WSRM1997]. However, for instance-intensive workflows, the scheduling algorithm is executed frequently, thus the execution cost of the scheduling algorithm also needs to be considered. The computation cost of Genetic Algorithms is very high due to iterations, thus not suitable for our scenario. On the contrary, the Min-Min algorithm gave excellent performance and had a very small scheduling execution time and is much more suitable for scheduling instance-intensive workflows. So we choose Min-Min as our base algorithm.

Before giving the steps of the EMM algorithm, we introduce two data structures which will be used for the algorithm. We depict them in Figure 3.2.

In workflow systems, resources are needed to execute tasks, and for every resource  $R_k$ , it can only be occupied by a task at one time. So we can allocate resources to tasks by assigning time slots to selected tasks. As shown on the top of Figure 3.2, when it is decided to schedule a task  $I_i T_j$  (i.e. the  $j^{\text{th}}$  task of the  $i^{\text{th}}$  workflow instance) on  $R_k$ , we will assign a suitable time slot of this resource to it. Of course, the data dependency and control dependency should be preserved simultaneously.

As shown at the bottom of Figure 3.2, the element  $e_{ij}$  in matrix  $e$  means the estimated execution time for task  $i$  on resource  $j$ . If its value equals to  $-1$ , it means that task  $i$  cannot be executed on resource  $j$ . The original values of this matrix are estimated, but we can modify the values at runtime according to the history execution record for better estimation later. The steps of the algorithm are described below:

- Select ready tasks from submitted instances: A ready task is the start task or a task such as whose predecessors all have been done if the ‘join’ condition of the task is ‘and’ or one of its predecessors has been done if the ‘join’ condition of the task is ‘or’. It should be noted that instances are traversed in order of their submittal time, which guarantees that tasks which came earlier have higher priority to be scheduled

than those came later. Thus the execution time of individual instances can be as minimal as possible.

- Compute the *EST* (earliest start time) on each capable resource for each task, while considering data dependency and control dependency. *EST* of a task is defined as follows. If the join condition of the task is “and”, the *EST* equals to the latest completion time of its predecessors; if the join condition of the task is “or”, the *EST* equals to the completion time of its earliest finished predecessor.
- Schedule the task with the minimum *EST* of all tasks to the resource which is expected to complete it at the earliest time. Then the algorithm will update the data of the corresponding resource.
- Go to the second step to schedule the remaining tasks until all ready tasks are scheduled.
- Wait until the next scheduling period comes. During this time, new instances can be added and some unready tasks may become ready because of the completion of certain tasks.

### **3.4.3 Complexity Analysis**

#### **3.4.3.1 Computation Complexity of OAL Algorithm**

For the OAL algorithm, first for the distance measure procedure, suppose the average number of neighbours is  $K$  and the maximum time that a message can be relayed is  $n$ , the procedure will simply involve a maximum of  $K^n$  nodes around the source node. However, after messages are sent from a node to its  $K$  neighbours, the neighbours will relay the messages in parallel. So this step will have the computation complexity of  $O(n)$ .

Then for the instance distribution procedure, the negotiation will also be conducted within these  $K^n$  nodes in parallel at a maximum of  $n$  rounds. So this step will also have the computation complexity of  $O(n)$ .

In summary, the computation complexity of the OAL algorithm is  $O(n)$ , where  $n$  is the maximum distance that a message can be relayed.

### 3.4.3.2 Computation Complexity of EMM Algorithm

For the EMM algorithm, first for the ready tasks selection step, the EMM algorithm checks each remaining workflow instance to find ready tasks. For each workflow instance, EMM algorithm checks each remaining task to find if it is ready. Suppose the total number of tasks in all workflow instances is  $T$ , this step will have the computation complexity of  $O(T)$ .

Next, for the EST (earliest start time) computation step, the EMM algorithm computes EST on each capable resource for each ready task. Suppose the average number of capable resource for each ready task is  $R$ , this step will have the computation complexity of  $O(T*R)$  if all tasks will be executed. Then, to find the task with the minimum EST of all tasks, the EMM algorithm will have the computation complexity of  $O(T)$  to find the task. Considering the loop for scheduling all tasks, this step will have the computation complexity of  $O(T^2*R)$ .

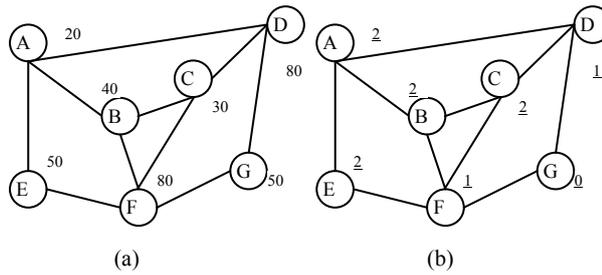
The scheduling step will have the computation complexity of  $O(1)$  to schedule a task to the suitable resource. Then the task repeats until all ready tasks are scheduled with a time complexity of  $O(T)$ .

In summary, with a maximum of  $T+T*(T*R+T)$ , the EMM algorithm will have the computation complexity of  $O(T^2*R)$ , where  $T$  is the number of all tasks, and  $R$  is the average number of capable resource for each task.

## 3.5 Example for Demonstration

### 3.5.1 Example for Demonstration of the OAL Algorithm

In this section, we address a simple example. Figure 3.3(a) shows a mesh made of super nodes  $A-G$ . The value marked for each node represents the current load value, Figure 3.3(b) shows the distance value of each node from node  $G$ .



**Figure 3.3 Simple Example of Mesh**

As defined in Section 3.4.1, for a specific node  $A$ , the  $OAL$  value of its neighbour  $B$  is the average load value of  $B$  and all its neighbours except  $A$ . Take the  $OAL$  value of Node  $B$  in Node  $A$  in Figure 3.2 for example:

$$OAL(B | A) = (Load(B) + Load(C) + Load(F))/3 = (40+30+80)/3 = 50$$

Table 3.1 shows the  $OAL$  values of every node in Figure 3.2. If  $OAL(X \text{ in } Y)$  equals to “-”, it means  $X$  and  $Y$  are not neighbours, and as a special case, the  $OAL(X \text{ in } X)$  is defined to  $Load(X)$ . Of course, the super node exchanges load values with each neighbour for calculation of  $OAL$  values periodically.

**Table 3.1 OAL Value Table**

	A	B	C	D	E	F	G
A	20	50	-	53	65	-	-
B	50	40	63	-	-	52	-
C	-	63	30	50	-	-	-
D	37	-	50	80	-	-	65
E	47	-	-	-	50	50	-
F	-	30	50	-	35	80	65
G	-	-	-	50	-	50	50

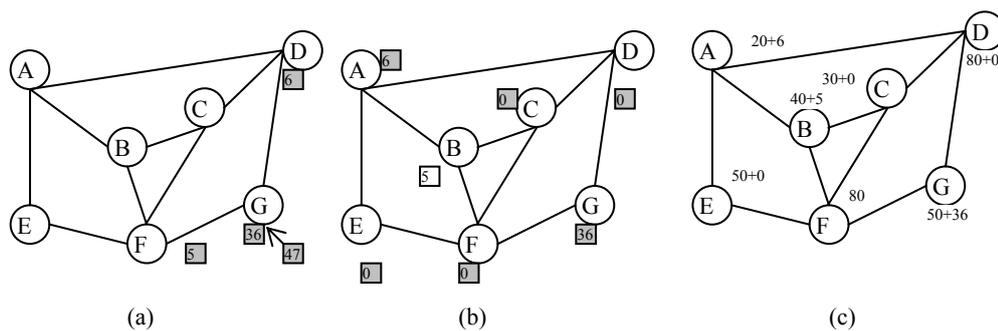
### 3.5.1.1 Scheduling Process of the Original Gossip Algorithm

Suppose 47 instances are submitted to node  $G$ . After applying the original Gossip algorithm, the scheduling steps can be described as follow:

In the first scheduling cycle, Node  $G$ , which has a load of 50, has two neighbours: Node  $D$  (load value: 80) and Node  $F$  (load value: 80). (See Figure 3.2). After it receives 47 instances, although the instances are indeed processed one by one, as a final result, it ends up that Node  $G$  keeps 36 instances itself and passes 6 instances to Node  $D$  and 5 instances to Node  $F$  for the purpose to balance the load in its neighbourhood. This is done to make the load value of all nodes in this area about the same. In this example, each node in the neighbourhood of Node  $G$  has a load value of about 86 after the procedure.

In the second scheduling cycle, after Node  $D$  receives the 6 instances from Node  $G$ , it will keep no instance itself and pass all 6 instances to the less loaded Node  $A$  (load value: 20) in its neighbourhood for the same purpose to balance the load in its neighbourhood. In another parallel procedure, After Node  $F$  receives the 5 instances from Node  $G$ , it will also keep no instance itself and pass all 5 instances to the less loaded Node  $B$  (load value: 40) in its neighbourhood.

In the third scheduling cycle, Node  $A$  and Node  $B$  will keep all instances passed to them because they are the least loaded node in its neighbourhood respectively. Thus the gossip procedure ends and the scheduling result is displayed in Figure 3.4. It can be seen clearly that this algorithm needs great improvement.



**Figure 3.4 Scheduling Result of Original Gossip Algorithm**

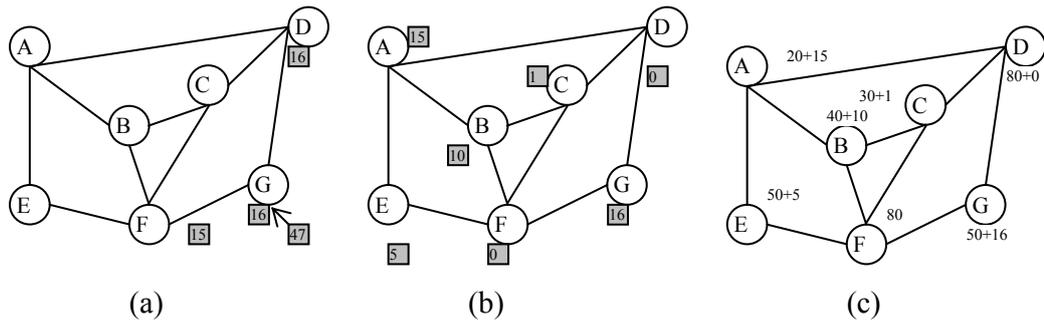
### 3.5.1.2 Scheduling Process of the OAL Algorithm

Suppose 47 instances are submitted to node  $G$ . After applying the OAL algorithm, the scheduling steps can be described as follow:

In the first scheduling cycle, Node G, which has a OAL value of 50, has two neighbours: Node D(OAL value: 50) and Node F(OAL value: 50). (See Figure 3.2). After it receives 47 instances, it will keep 16 instances itself and pass 16 instances to Node D and 15 instances to Node F for the purpose to balance the load in its neighbourhood,. This is done to make the OAL value of all nodes in this area about the same. In this example, each node in the neighbourhood of Node G has an OAL value of about 66 after the procedure.

In the second scheduling cycle, after Node D receives the 16 instances from Node G, it will keep 0 instances itself and pass 15 instances to Node A(OAL value: 37) and 1 instances to Node C(OAL value: 50) for the same purpose to balance the OAL in its neighbourhood. In another parallel procedure, After Node F receives the 15 instances from Node G, it will keep 0 instances itself and pass 10 instances to Node B(OAL value:30), 0 instance to Node C(OAL value: 50) and 5 instances to Node E(OAL value: 35).

In the third scheduling cycle, the procedure will end because there are no further nodes with distance greater than 2 existing to distribute on. The scheduling result of the OAL algorithm is shown in Figure 3.5. It can be seen that the overall load is much more balanced than that of the original Gossip algorithm.

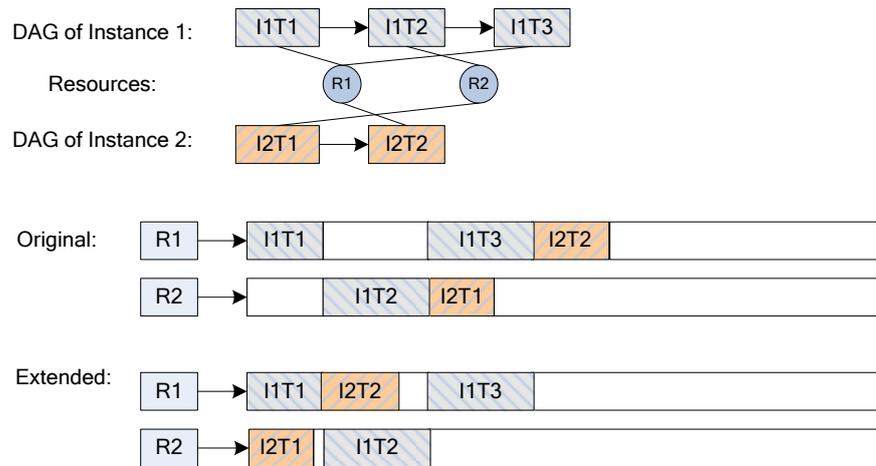


**Figure 3.5 Scheduling Result of OAL Algorithm**

### 3.5.2 Example for Demonstration of the EMM Algorithm

Compared with the original Min-Min algorithm, the EMM algorithm has its particular advantages. We can use a simple case to demonstrate the advantages of the usage of time slots. Suppose there are two tasks which need to be scheduled concurrently, we will analyse the two algorithms as follows. The Min-Min algorithm uses a Completion

Matrix for the calculation of *ECT* (Earliest Completion Time), and once a task has been allocated to a resource, the *ECT* will be updated accordingly. After instance *I* has been finished, the next available time of resource 2 will be set to the completion time of  $I_1T_2$ . Then instance 2 will be scheduled based on the new matrix. The scheduling results are shown in Figure 3.6. For the EMM algorithm, the time slots before  $I_1T_2$  are still available after instance *I* has been scheduled and can be allocated to the first task of instance 2 for execution.



**Figure 3.6 Example of Scheduling Results of Original and Extended Min-Min Algorithms**

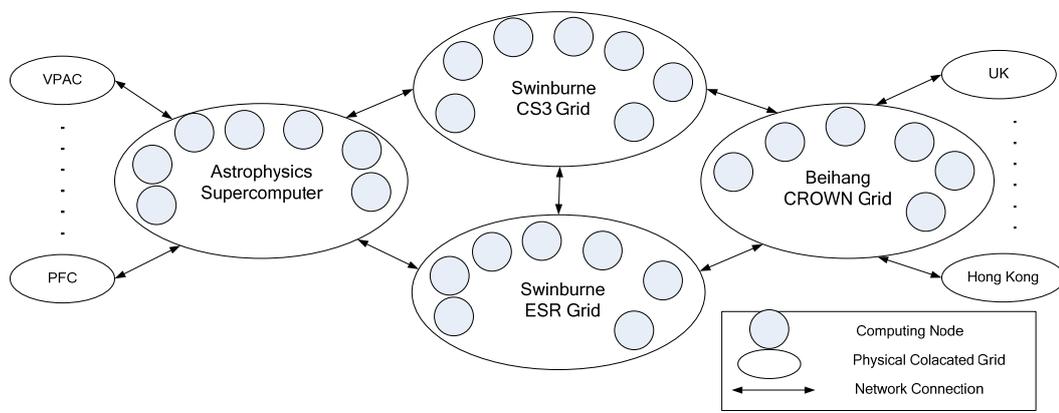
### 3.6 Comparison and Simulation

In this section, we will perform an experimental simulation in the Swinburne Workflow Test Environment. Our aim is to build the scheduling infrastructure depicted in Section 3.3 (denoted as SwinDeW-Cg: **S**winburne **D**ecentralised **W**orkflow for **C**loud based on **G**rid) on top of it and perform the original Gossip algorithm, the OAL algorithm, the original Min-Min algorithm and the EMM algorithm on the scheduling infrastructure in order to demonstrate the advantages of the OAL algorithm and the EMM algorithm.

#### 3.6.1 Simulation Environment

The Swinburne Workflow Test Environment is depicted in Figure 3.7. It contains many grids distributed in different places all around the world. Each grid node contains many computers including high performance PCs and/or supercomputers composed of

significant number of computing units. The primary hosting nodes include the Swinburne CS3 (Centre for Complex Software Systems and Services) Grid, Swinburne ESR (Enterprise Systems Research laboratory) Grid, Swinburne Astrophysics Supercomputer Grid, and Beihang CROWN (China R&D environment Over Wide-area Network) Grid in China. Furthermore, the CROWN Node is also connected to some other grids such as those in Hong Kong University of Science and Technology and University of Leeds in UK. The Swinburne Astrophysics Supercomputer Grid is cooperating with such as PfC (Australian Platform for Collaboration), VPAC (Victorian Partnership for Advanced Computing) and so on.

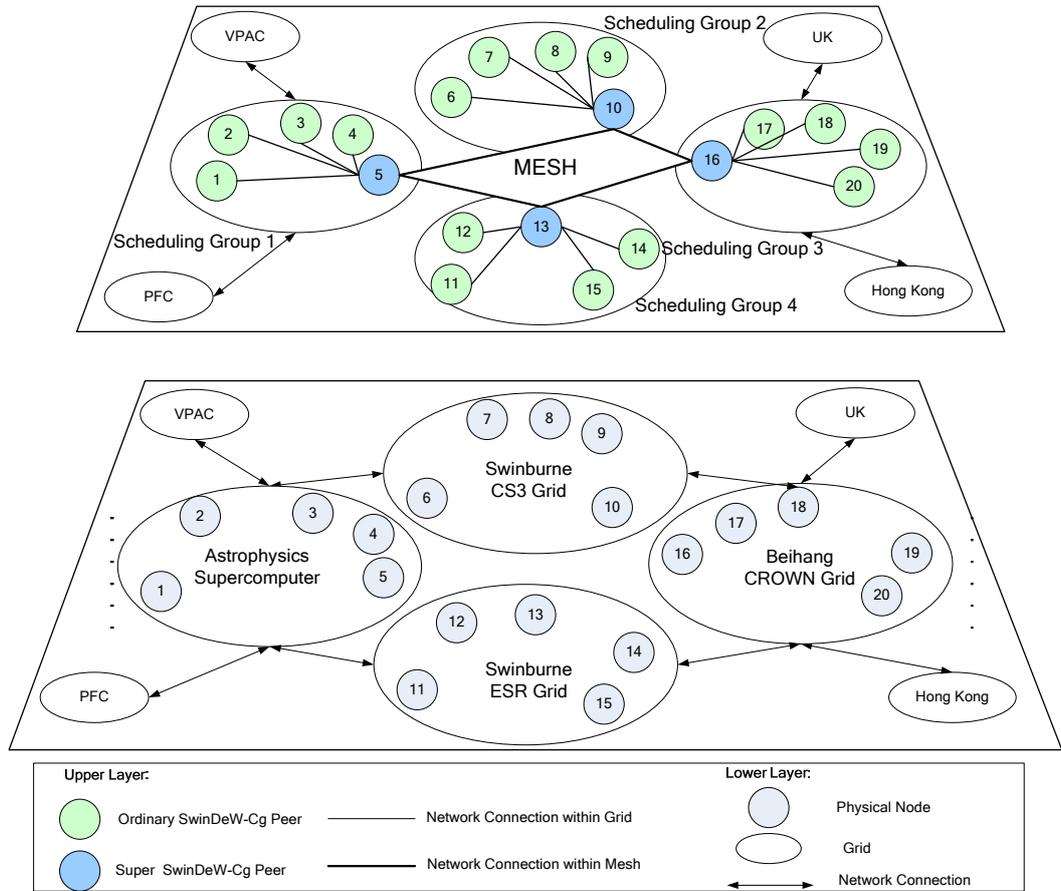


**Figure 3.7 Swinburne Workflow Test Environment (SWTE)**

As shown in Figure 3.8, SwinDeW-Cg is built on top of the Swinburne Workflow Test Environment (SWTE). The upper layer represents the logic infrastructure of scheduling algorithm, while the lower layer represents the SWTE. The numbers in the figure represent the mapping relationship of logic scheduling peers and physical nodes, that is, if a number on a logic scheduling peer on the upper layer is the same with that of a physical node on lower layer, it implies that this logic scheduling peer is implemented on this physical node.

It can be seen from Figure 3.8 that each physical grid on the lower layer is organised naturally as an autonomous scheduling cloud of the scheduling algorithm on the upper layer, and each grid has a designated node as the super node, while all super nodes constitute a mesh on the upper layer.

The architecture of SwinDeW-Cg is shown in Figure 3.8 with SWTE listed on the bottom for reference.



**Figure 3.8. Swinburne Decentralised Workflow for Cloud based on Grid (SwinDeW-Cg)**

### 3.6.2 Criteria for Comparison

#### 3.6.2.1 Criteria for Comparison between the Original Gossip Algorithm and the OAL Algorithm

We compare the original Gossip algorithm and the OAL algorithm based on the *variance deviation*. The variance deviation of the loads of all scheduling clouds can be defined as follows.

$$\sigma^2 = \frac{\sum(x - \bar{x})^2}{n}$$

Where  $\bar{x}$  is the mean load value,  $n$  is the number of groups, and  $x$  stands for each load value of each group in turn. This value represents how far the observations deviate from the mean. The smaller this value is, the more balanced the system is.

### 3.6.2.2 Criteria for Comparison between the Original Min-Min Algorithm and EMM Algorithm

As for the EMM algorithm, we use about 10,000 different workflow instances which contain roughly 100,000 tasks to test the performance. The comparison criteria include the *mean execution time*, the *resource utilisation rate* and the *overall throughput*.

- *Mean execution time* is the average execution time of all workflow instances.
- *Resource utilisation rate* is the percentage of working time of resources in a period of time.
- *Overall throughput* is measured as the number of workflow instances completed in a second.

Obviously, the shorter the *mean execution time* is, the higher the *resource utilisation rate* is; and the higher the *overall throughput* is, the better the algorithm is.

### 3.6.3 Simulation Process

As mentioned in Section 1.1, instance-intensive cloud workflows are *workflows with a huge number of (hence instance-intensive) concurrent workflow instances, usually much simpler than those complex scientific workflows, enabled on a cloud computing environment (hence cloud workflows)*. To simulate such workflows on grid-based cloud computing environment, we construct the simulation environment focusing on the following three aspects:

1. We have initialised a database which includes all kinds of DAGs that represent instance-intensive workflows. Each of these workflows is a simple workflow with only 5-10 tasks, and the duration of each task is 0.1-1 second. A small proportion of them are generated in advance from practical workflows such as bank cheque processing, while others are randomly generated by software.
2. In order to simulate the huge number of concurrent instances, we use a workflow instance generator which creates 1,000 instances per second. Each instance is

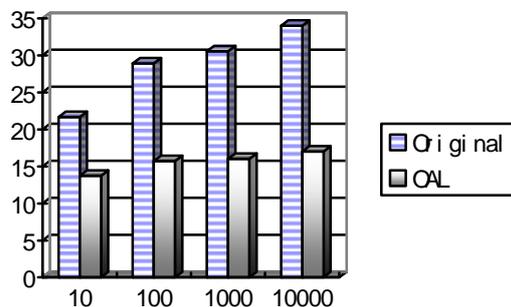
generated from a randomly selected DAG from the DAG database. The reason for the generation rate of 1,000 instances per second is due to the computation power of the simulation environment. However, via defining one second in real world equals to  $n$  seconds in simulation, this generation rate can be amplified to fulfil the simulation purpose.

3. For the OAL algorithm, we set the initial load of every group to a random value, so the initial overall load is usually unbalanced.

In the simulation, we first implement the OAL algorithm and the original Gossip algorithm on SwinDeW-Cg, then implement the extended Min-Min algorithm and the original Min-Min algorithm, and finally run the simulation and compare their performance according to the criteria as described in Section 3.6.2.

### 3.6.4 Results and Analysis

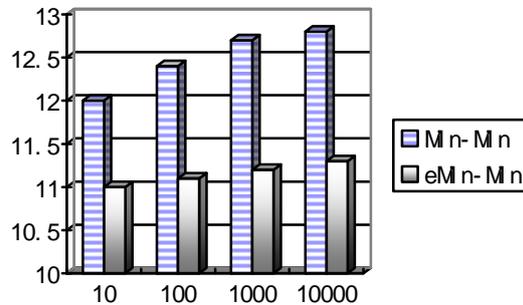
Figure 3.9 shows the variance deviation of load values of the original Gossip algorithm and the OAL algorithm. The horizontal axis is the number of instances we put into simulation, and the vertical axis is variance deviation of load values of the original Gossip and OAL algorithms. We can see that the variance deviation of load values of the OAL algorithms is much smaller than that of the original Gossip algorithm, which shows the former algorithm can produce more balanced scheduling results.



**Figure 3.9 Variance Deviation of Load Values of Original Gossip and OAL Algorithms**

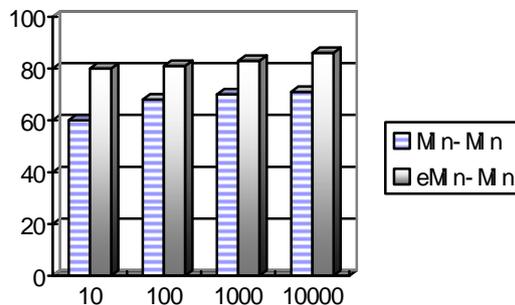
In Figure 3.10, the horizontal axis is the number of instances we put into simulation, and the vertical axis is the mean execution time of all instances. We can see that the

mean execution time of the EMM algorithm is less than that of the original Min-Min algorithm.



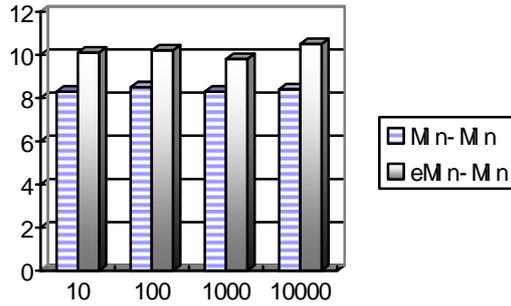
**Figure 3.10 Mean Execution Time of Original and Extended Min–Min Algorithms**

In Figure 3.11, the horizontal axis is the number of instances we put into simulation. The vertical axis is the average resource utilisation rate in percentage of all resources. It can be seen that the resource utilisation rate of the extended Min-Min algorithm is higher than that of the original Min-Min algorithm.



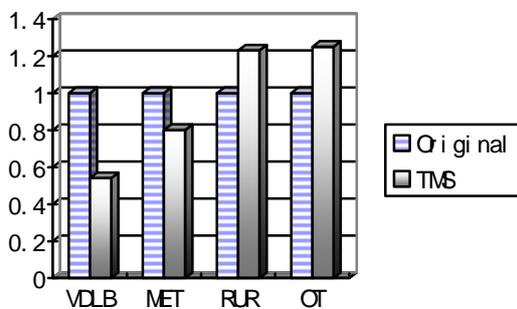
**Figure 3.11 Resource Utilisation Rate of Original and Extended Min–Min Algorithms**

In Figure 3.12, the horizontal axis is the number of instances we put into simulation, and the vertical axis is the number of instances finished per millisecond. When it comes to the overall throughput in a certain time, it can be seen from Figure 3.12 that the overall throughput of the extended Min-Min algorithm is higher than that of the original Min-Min algorithm.



**Figure 3.12 Overall Throughput of Original and Extended Min–Min Algorithms**

To present an overall comparison based on the above analysis, in Figure 3.13 we summarise the overall advantage of the proposed Throughput Maximisation Strategy from all aspects mentioned above. In this figure, *VDLB*, *MET*, *RUR* and *OT* represent the *variance deviation of load values*, the *mean execution time*, the *resource utilisation rate* and the *overall throughput* respectively. For each aspect, we assume the performance of the original algorithm is 1 and compute the relative performance of the proposed Throughput Maximisation Strategy accordingly.



**Figure 3.13 Overall Advantage of Proposed Throughput Maximisation Strategy (TMS)**

From Figure 3.13, we can tell that the OAL algorithm is about twice as balanced as the original Gossip algorithm, and the EMM algorithm reduces approximately 20% less execution time and has roughly 20% higher resource utilisation rate than the original Min-Min algorithm, thus has about 25% higher overall throughput than the original Min-Min algorithm.

### 3.7 Summary

As mentioned in Section 1.2, the most common infrastructures of cloud computing environment consists of physically collocated grids (grid-based), cloud computing environment consists of free services scattered on the Internet (service-based) and cloud computing environment consists of charged services (business-based). This chapter mainly focus on scheduling instance-intensive workflows on grid-based cloud computing platform, scheduling on service-based cloud computing platform will be discussed in Chapter 4, and scheduling on business-based cloud computing platform will be discussed in Chapter 5.

To efficiently schedule instance-intensive workflows on grid-based cloud computing platform, we have proposed a Throughput Maximisation Strategy (TMS) which contains two specific scheduling algorithms on different two layers for scheduling instance-intensive workflows. The two algorithms are the OAL (Opposite Average Load) algorithm for instance scheduling and the EMM (extended Min-Min algorithm) algorithm for task scheduling. The first one called Opposite Average Load algorithm (OAL) tries to maximise the overall throughput in order to build a strong basis for the second step by pursuing overall load balance at instance level, while the second one called Extended Min-Min algorithm (EMM) dedicates to further maximise the overall throughput at task level by increasing the utilising rate of resources within each autonomous group. We have explained in Section 3.4 why these two algorithms are better than their respective original algorithms in theory. The comparison and simulation in Section 3.6 have practically demonstrated the two algorithms can produce better scheduling results than their respective original algorithms in practice when it comes to the scheduling of instance-intensive workflows on grid-based cloud computing platform.

# Chapter 4

## Min-Min Average Algorithm for Scheduling Instance-intensive Cloud Workflows Involving Communication Overhead

This chapter is based on our paper [LCJY2009] and is organised as follows. In Section 4.1, a brief introduction to the background of the proposed algorithm is given, followed by the requirements analysis of scheduling for instance-intensive workflows in a grid-based cloud computing environment in Section 4.2. Section 4.3 presents the fundamental design of the proposed algorithm: Min-Min-Average algorithm, and Section 4.4 discusses the details of the MMA algorithm. Later Section 4.5 gives examples to illustrate the advantages of the MMA algorithm and Section 4.6 demonstrates the benefits of our work via simulation and comparison, and finally in Section 4.7, we summarise this chapter.

### 4.1 Introduction

As mentioned before, some e-business applications such as a bank cheque processing application may need to process millions of transactions per day (hence being referred to as transactions intensive), whilst some of which may need considerable data transfer capabilities (e.g. for images) among different tasks to fulfil transactions. Hence they can be modelled as instance-intensive cloud workflows with considerable communication overheads. The main characteristic of such workflows is a huge number of relatively simple concurrent instances of which may involve considerable communication overheads among relevant tasks. Thus it is necessary to consider multiple instances of

multiple workflows with communication overheads when designing scheduling algorithms for such workflows.

Again, although there exist many workflow scheduling algorithms, most of them are proposed for scheduling computation-intensive scientific workflows. Scheduling algorithms are rarely dedicated for instance-intensive workflows and almost none are designed for instance-intensive workflows taking communication overheads into account. It is emergent to design corresponding algorithms for workflows of this nature.

In this chapter we focus on scheduling instance-intensive workflows on service-based cloud computing platform, scheduling on grid-based cloud computing platform has already been discussed in Chapter 3, and scheduling on business-based cloud computing platform will be discussed in Chapter 5.

## **4.2 Requirements Analysis and Overall Solution**

### **4.2.1 Requirements Analysis**

Different from conventional computation-intensive workflow scheduling algorithms, algorithms designed for instance-intensive cloud workflows have the following primary requirement:

- Due to a huge number of concurrent workflow instances, the algorithms should focus on minimising the mean execution time of all instances in order to maximise the overall throughput rather than minimising the execution time of individual instances.

When considerable communication overheads are taken into account, instance-intensive workflow scheduling algorithms should also consider the following requirements:

- The communication bandwidth, delay and so forth are constantly changing on the Internet, thus dynamic adaptation to network change should be reflected in the algorithms;

- The communication time can no longer be ignored, and thus should be considered as another important factor along with execution time during algorithm design.

### **4.2.2 Overall Solution**

To meet the requirements mentioned above, we propose a Min-Min-Average algorithm (MMA) for scheduling instance-intensive workflows involving considerable communication overheads on a service-based cloud environment.

First, we establish a fundamental design for the algorithm that can provide nearest neighbours, i.e., the nodes that have the shortest transmission delay to the specific node for joint scheduling planning. The neighbours are maintained based on real-time probing in order to adapt to the network circumstance automatically. Due to this adaptation, the communication time can be decreased significantly, which is an important contributing factor for high overall throughput of the MMA algorithm.

Second, because the communication time can no longer be ignored, we develop a decision making approach for MMA which takes communication time into account for allocating tasks to appropriate resources.

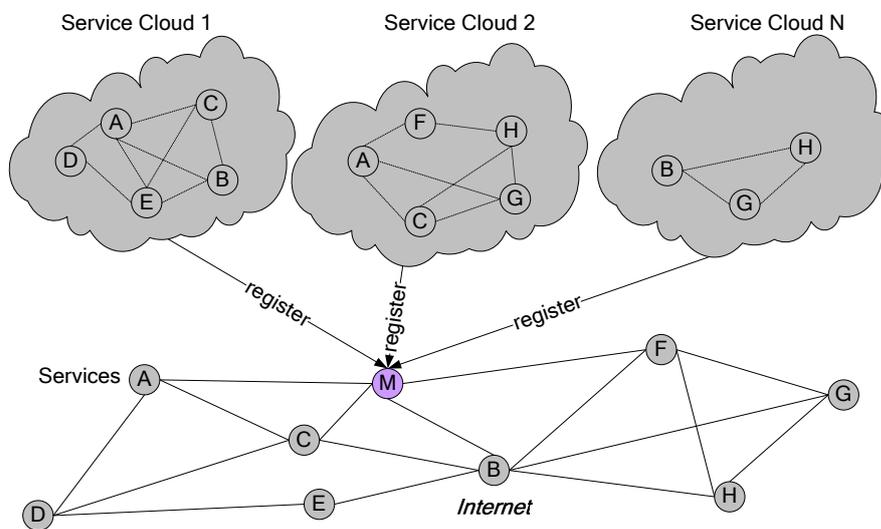
Third, due to the importance of both execution time and transmission time, we propose a different strategy to the original Min-Min algorithm [MASH+1999] which can increase the utilisation rate of both the execution unit and transmission unit of workflow execution in order to reduce the mean execution time of all instances for increasing overall throughput.

## **4.3 Scheduling Infrastructure**

In this section, we demonstrate the scheduling infrastructure of in order to propose our MMA scheduling algorithm. First we introduce some concepts of service clouds, followed by data structures of ordinary node and monitor node. Finally, we describe how to maintain the neighbours and find a set of nearest nodes for a specific resource.

### 4.3.1 Service cloud

As Figure 4.1 shows, the workflow execution environment is divided into several service clouds according to different services available. For each new computing node, it joins the service clouds according to individual services it provides, such as scanner service and printer service. If the service cloud already exists, the node simply joins it; otherwise, it creates a new service cloud and joins it as a creator. There are two kinds of nodes, namely, ordinary and monitor nodes as described later in this section. It should be noted that all service clouds are registered and maintained by the monitor nodes.



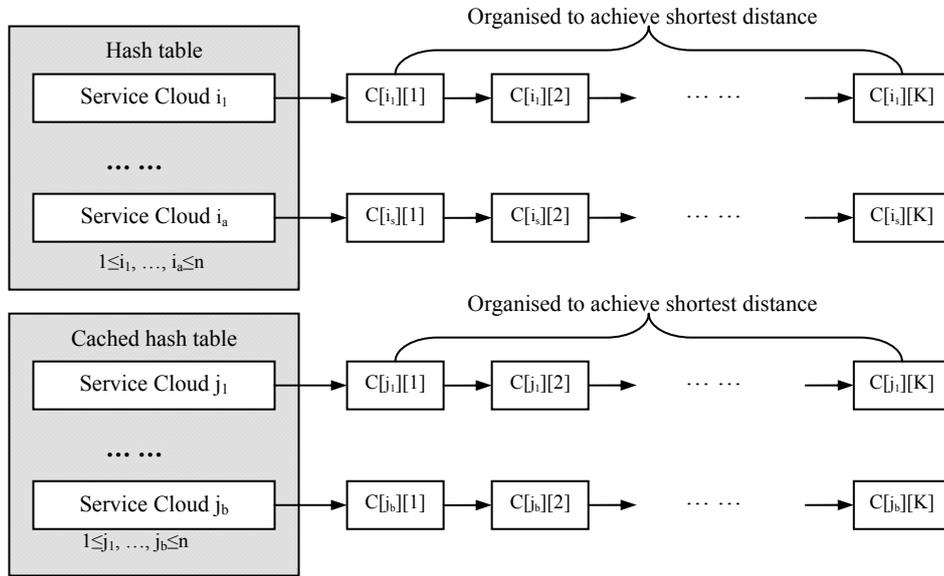
**Figure 4.1 Scheduling Infrastructure of MMA**

As we can see in Figure 4.1, node *M* indicates that it is a monitor node, managing each service cloud (one per cloud). Ordinary nodes like *D* and *E* have service *1*, thus join service cloud *1* only. Meanwhile, ordinary nodes *A* and *C* both have resources *1* and *2*, thus join service clouds *1* and *2*, whilst ordinary nodes *G* and *H* both have resources *2* and *N*, thus join service clouds *2* and *N* accordingly.

### 4.3.2 Ordinary Node

For the purpose of efficiently scheduling instance-intensive cloud workflows with considerable communication overheads, every ordinary node maintains its nearest nodes called neighbours in each service cloud it joins. As Figure 4.2 shows, there are two hash tables on an ordinary node. The top hash table contains the service clouds the node joins

and the bottom hash table caches the most recently used service clouds the node does not join. For each service cloud, it maintains a list of neighbours and maximum  $K$  ( $K$  is set to 35 by default as in BitTorrent [BCCM+2006]), to achieve the shortest distance which can be measured with RTT (Round Trip Time).

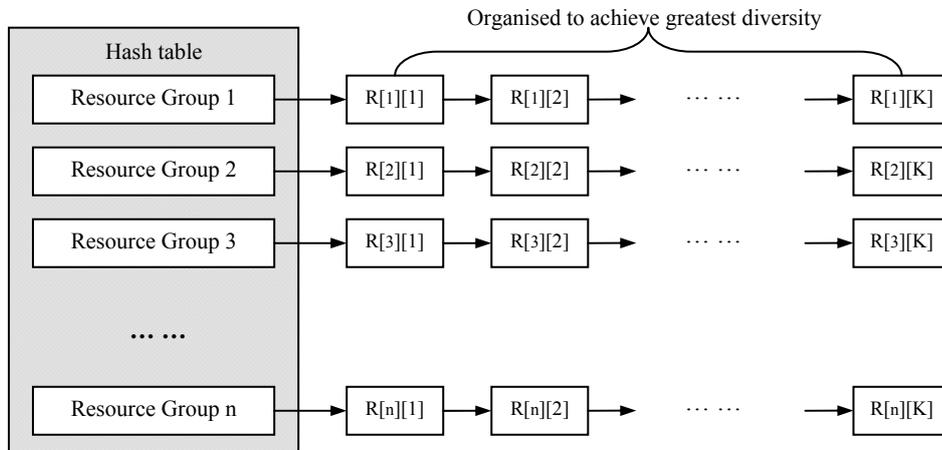


**Figure 4.2 Data Structure of Ordinary Node**

In order to maintain the neighbour list, a probing procedure is performed periodically. In each period, every host node tests the network speed with a number of randomly selected neighbours (the number of selected neighbours is usually relatively small compared with the total number of neighbours) and compares the value with candidate neighbours. A feasible selection strategy is to divide the neighbours in the list into  $n$  small groups due to their index, for example, the neighbours whose indices mod  $n = 0, 1, \dots, n-1$  are grouped together respectively and selected by the probing procedure in turn. The candidate neighbours include recently contacted neighbours and candidate neighbours exchanged with other nodes. An old neighbour will be replaced by a new neighbour that is nearer. The whole maintenance cost is linear, with a small factor, to the total number of its neighbours and usually negligible.

As we mentioned before in Section 4.2.1, this structure is designed to adapt to the fluctuating delay on the Internet. For keeping a group of nearest neighbours for each service cloud, the data transfer time between two successive tasks is guaranteed to be as short as possible.

### 4.3.3 Monitor Node



**Figure 4.3 Data Structure of Monitor Node**

As Figure 4.3 shows, there is a hash table which contains all service clouds on the monitor node. For each service cloud, it also maintains a list of neighbours and maximum  $K$ . The neighbour choosing strategy of the monitor node can have a significant impact on the scheduling performance. It is often the case that a list of geographically distributed nodes provides much greater utility than that clustered together.

The diversity can be achieved by the periodical checking procedure which is similar to that of ordinary nodes. The difference between them is that a monitor node reassesses neighbours in each service cloud and replaces some neighbours with alternatives that provide greater diversity, rather than nearer neighbours in an ordinary node. Similar to the maintenance procedure of the ordinary nodes, the complexity of the maintenance cost of the monitor node is also linear, with a small factor, to the total number of its neighbours and usually negligible.

### 4.3.4 Node Maintenance

The join procedure to a service cloud can be briefly described as follows. First, the joining node will search if the service cloud exists on the monitor node. If it exists, the neighbour list linked to the service cloud is traversed to find maximum  $m$  nearest nodes and returned to the joining node. Second, the joining node will request maximum  $n$

nearest nodes from each node of the  $m$  nearest nodes, then select  $k$  nearest nodes from these maximum  $m*n$  nodes. If this service cloud does not exist, the joining node will create the service cloud itself and joins it as the creator.

In addition, heartbeat messages are used for the detection of leaving nodes, and the neighbour information adhered to the heartbeat messages can be used for candidate neighbour exchange, which is usually used for neighbour list update. It should be addressed that the neighbour exchange operation is conducted jointly with the heartbeat message, thus no additional cost is involved.

#### **4.3.5 Node Search**

The purpose of node search is to find a number of capable nodes that have a specific resource needed to execute a task. The scheduling algorithm needs these nodes for joint planning. If the host node itself is in the service cloud of the required resource, it is easy to find a set of nearest capable neighbours from the neighbour list due to the neighbour organisation method. As for other resources that the node does not have the neighbour list, the cached hash table will be searched first. If the service cloud can be found in the hash table, the nearest capable neighbours are returned from the neighbour list directly; otherwise, the monitor node will be facilitated to find a set of nearest nodes with the specific resource.

The search procedure for a list of nodes with a specific resource via the monitor node is described as follows. First, the hash table of the monitor node is searched to find out whether the service cloud of the specific resource exists. If the service cloud exists, the neighbour list is traversed to find maximum  $m$  nearest nodes from the query node which is the node that invoked the query. Second, the query node requests maximum  $n$  nearest nodes from each node of the  $m$  nearest nodes. Finally,  $k$  nearest nodes are selected from these maximum  $m*n$  nodes.

#### **4.4 MMA Scheduling Algorithm**

In this section, we detail the MMA algorithm. In the MMA algorithm, the scheduling is performed at the task level. Every node can perform scheduling independently whilst

the scheduling is conducted jointly with its nearest neighbours. In fact, the MMA algorithm runs on every node periodically for scheduling workflow tasks which are submitted to the node. In this section, we take one node as an example to demonstrate how the algorithm works, and for the purpose of convenience, we refer this selected node as the host node.

#### 4.4.1 Terms Used in MMA Algorithm

Before addressing MMA, we introduce the terms used in the MMA algorithm. The MMA algorithm is derived from the popular Min-Min algorithm [MASH+1999], thus some terms can be borrowed from the Min-Min algorithm. We introduce the related terms defined in the Min-Min algorithm first and then new terms used in the MMA algorithm.

The related terms defined in the Min-Min algorithm are listed below:

- $EET(r, t)$ : It is defined as the estimated execution time of task  $t$  at resource  $r$ .
- $EAT(r, t)$ : It is the estimated time at which resource  $r$  will become available to perform task  $t$ , i.e. the time at which it will have finished executing all the tasks before  $t$  in its queue.
- $DAT(r, t)$ : It is defined as the earliest time by which all the data/files required by task  $t$  are available at resource  $r$ . It is related to the input data size of task  $t$  and the network transmission speed, which represents the communication overheads mentioned in this chapter.
- $ECT(r, t)$ : It is the estimated completion time at which task  $t$  would be completed at resource  $r$ . We have  $ECT(r, t) = EET(r, t) + \max(EAT(r, t), DAT(r, t))$ .

In order to describe the MMA algorithm, we introduce the following new terms:

- $OTAT(r)$ : It is the time at which the resource will be available for execution of tasks in this scheduling round, i.e. the time at which it will have finished executing all the tasks of previous scheduling rounds in its queue.
- $OEAT(r)$ : It is the time at which the resource will be available for data transmission in this scheduling round, i.e. the time at which it will have finished transmitting all data required by tasks of previous scheduling rounds.
- $TRAN(r, t)$ : It is the transmission time of the data required by task  $t$  to resource  $r$ .
- $ETAT(r, t)$ : It is the estimated data transfer unit available time of resource  $r$  after task  $t$  has been scheduled on the resource. It can be calculated recursively by the equation:  $ETAT(r, t) = ETAT(r, t-1) + TRAN(r, t)$ . In particular,  $ETAT(r, 0) = OTAT(r)$ .
- $EEAT(r, t)$ : It is the new estimated execution unit available time of resource  $r$  after task  $t$  has been scheduled on the resource. We have  $EEAT(r, t) = EET(r, t) + \max(TRAN(r, t), ETAT(r, t))$ . In particular,  $EEAT(r, 0) = OEAT(r)$ .

#### 4.4.2 Overview of MMA

In order to give an overall picture of the MMA algorithm, as Table 4.1 shows, the MMA algorithm can be divided into the following steps. The details will be described in Section 4.4.3.

**Table 4.1: Pseudo Code of MMA Algorithm**

<p><b>Algorithm:</b> Min-Min-Average Scheduling Algorithm  <b>Input:</b> An array of workflow instances Instances[]  <b>Output:</b> A schedule</p>
<pre> <b>01 Procedure</b> Schedule(InstanceArray Instances[])     // Step 1: select ready tasks from the multiple instances of multiple workflows.     <b>02 for</b> each instance I in Instances [] <b>do</b>     <b>03 for</b> each task t in I <b>do</b>     <b>04 if</b> t is ready <b>then</b>     <b>05</b>     add t to ReadyTasks[];     <b>06</b>     <b>end if</b>     <b>07</b>     <b>end for</b> </pre>

```

08 end for
    // divide ready tasks to task groups against required resources;
09 for each task t in ready tasks do
10     If t requires resource r then
11         add t to task group that requires resource r;
12     end if
13 end for
    //Step 2: for each task group, select a list of capable nodes with requested resources.
14 for each task group tg do
15     if the host has the resource to execute the tasks in tg then
16         select a list of capable nodes with required resources nl(tg) from hash table in
Figure 2;
17     else if nodes can be found in the cached hash table then
18         select a list of capable nodes with required resources nl(tg) from cached hash
table;
19     else
20         request a list of capable nodes from the monitor node;
21     end if
22 end if
21 end for
    // for all selected neighbours, send request messages and wait for responses.
22 for each capable neighbour do
23     send a schedule-request message;
24 end for
    // wait for receiving response messages which contains OEAT and OTAT, and add
the sender to the node list of the corresponding task group.
25 wait for response and initialise node list nl(tg) for each tg;
26 for each response msg do
27     n = sender of the msg;
28     r = resource that node n has;
29     tg = the task group which need resource r to execute;
30     add node n to nl(tg);
31     set n.OEAT and n.OTAT to the values contained in the message;
32 end for
    //Step 3: schedule tasks group by group
33 for each task group tg in task groups do
34     repeat
        // find resource for each ready task with minimum AVG-EEAT-ETAT
35     MIN-AVG-EEAT-ETAT(t) = MAX_VALUE;
36     for each capable node with resource r in nl(tg) do
37         for each task t in tg do

```

```

38   calculate AVG-EEAT-ETAT(r, t);
39   if (AVG-EEAT-ETAT(r, t) < MIN-AVG-EEAT-ETAT(t)) then
40       MIN-AVG-EEAT-ETAT(t) = AVG-EEAT-ETAT(r, t);
41       PreferredResource(t) = r;
42   end if
43 end for
44 end for
    // find task with minimum MIN-AVG-EAT-ETAT of all tasks
45 MIN-MIN-AVG-EEAT-ETAT = MAX_VALUE;
46 for each task t in rg do
47     if (MIN-AVG-EEAT-ETAT(t) < MIN-MIN-AVG-EEAT-ETAT) then
48         MIN-AVG-EEAT-ETAT(t) = AVG-EEAT-ETAT(r, t);
49         next = t;
50     end if
51 end for
    // schedule next task to the resource that gives the MIN-MIN-AVG-EAT-ETAT
value.
52   schedule next to PreferredResource(next);
53   modify resource Information accordingly;
54   remove next from task group tg;
55   until all tasks in task group tq have been scheduled
56 end for
    //Step 4: distribute task data for execution, and then wait for the next scheduling
round.
57   distribute task t to PreferredResource(t);
58   sleep until next scheduling round comes;
59 end Schedule

```

### 4.4.3 Details of MMA

The details of each step of the MMA algorithm can be described as follow:

#### Step 1: Selection of Ready Tasks

The first step of MMA is to select ready tasks for scheduling. Ready tasks are selected from the multiple instances of multiple workflows by the host node. A ready task is the start task of a workflow instance or a task whose predecessors have all been finished.

The ready tasks with the same resource requirements are grouped and scheduled together, while different task groups can be scheduled in parallel.

## **Step 2: Selection of Nodes and Resources**

After ready tasks have been selected and grouped as task groups, for each task group, the host node searches for the nodes with the resources needed to execute the tasks in the group. Here we have three cases:

- The first case is that the host node itself has the required resource to execute the task, which also means that it is a member of the service cloud and has the neighbour information of the service cloud. In this case the neighbour list can be simply initialised from the neighbour list linked to the service cloud in the hash table, as shown in Figure 4.2.
- The second case is that the host node has no requested resource, but the neighbour information can be found in the cached hash table. In this case the neighbour list can also be initialised from the neighbour list of the service cloud in the cached hash table, as shown in Figure 4.2.
- The last case is that the host node does not have any information about the nodes with the specific resource. In this case it has to get the neighbour information with the help of the monitor node. The details of the search procedure have already been described in Section 4.3.5.

After such nodes are selected, the host node will send a scheduling request to each selected node. For the requested node, after receiving this message, if the node has not joined any other scheduling procedure, it will respond to the host node with a confirmation message which contains two parameters critical for scheduling instance-intensive workflows with considerable communication overheads:  $OTAT(r)$  and  $OEAT(r)$ .

After sending confirmation message in order to avoid resource conflicts, the neighbour will not respond to any other scheduling request until this scheduling round has been completed.

### **Step 3: Scheduling of Tasks**

This step is to map tasks onto resources. For each resource  $r$ , we can use two time-slot windows to represent the time allocation status of its execution unit and transmission unit respectively. The first window represents the time used for task execution on the resource and the second window stands for the time used for transferring the data needed for task execution to the resource, which is used to represent the communication overheads.

Considering the importance of both execution time and data transfer time for instance-intensive cloud workflows, we can improve the Min-Min local selection heuristic approach by using a different strategy, namely, the Min-Min-Average (MAA) algorithm to increase the utilisation rate of both execution unit and transmission unit of the resource. This step can be further divided into the following 3 sub-steps:

(1) For each task  $t$ , we calculate the average value of  $EEAT(r, t)$  and  $ETAT(r, t)$  on each resource (denoted as  $AVG-EEAT-ETAT(r, t) = (EEAT(r, t) + ETAT(r, t))/2$ ) and select the resource which provides the minimum average value of  $AVG-EEAT-ETAT(r, t)$  (denoted as  $MIN-AVG-EEAT-ETAT(r, t)$ ).

(2) The task which has the minimum  $MIN-AVG-EEAT-ETAT(r, t)$  among all tasks is scheduled to resource  $r$  that gives this value.  $EEAT(r, t)$  and  $ETAT(r, t)$  of resource  $r$  are updated correspondingly.

(3) The algorithm continues scheduling the remaining tasks in the ready tasks list until all tasks are scheduled in this round. It should be noted that values mentioned in previous sub-steps of all tasks will be recalculated based on the new status of resources.

#### Step 4: Distribution of Task Data

The final step is to distribute tasks to the nodes on which they will be executed. After all ready tasks being allocated to specific resources, the host node schedules each task to the respective neighbour by sending the definition and the input data of the task. Then the scheduler will sleep until the next scheduling round begins.

#### 4.4.4 Complexity Analysis

For the MMA algorithm, first for the ready tasks selection step, similar to the EMM algorithm, the MMA algorithm checks each remaining task of each remaining workflow instance to find if it is ready. Suppose the total number of tasks in all workflow instances is  $T$ , this step will have the computation complexity of  $O(T)$ .

Next, for the nodes and resources selection step, there are three cases.

- In the first case when the host node itself has the required resource to execute the task, the neighbour list can be simply initialised from the neighbour list linked to the service cloud in the hash table. Because the length of neighbour list is a constant not larger than  $K$  as described in Section 4.3.2, this step will have the computation complexity of  $O(1)$ .
- In the second case when the host node has no requested resource, but the neighbour information can be found in the cached hash table. Similar to the first case, because the length of cache neighbour list is also a constant not larger than  $K$  as described in Section 4.3.2, this step will have the computation complexity of  $O(1)$ .
- For the last case when the host node does not have any information about the nodes with the specific resource, it has to get the neighbour information with the help of the monitor node. The details of such a procedure are described as follow. First, the hash table of the monitor node is searched to find the service cloud of the specific resource. This will have the computation complexity of  $O(L)$ , where  $L$  is the length of the hash table on the monitor node. If the service

cloud exists, the neighbour list is traversed to find maximum  $m$  nearest nodes from the query node which is the node that invoked the query. This will also have the computation complexity of  $O(L)$ . Second, the query node requests maximum  $n$  nearest nodes from each node of the  $m$  nearest nodes. Because the requests are issued in parallel, this step have the computation complexity of  $O(I)$ . Finally,  $k$  nearest nodes are selected from these maximum  $m*n$  nodes. Because  $m$  and  $n$  are both constants, this step will have the computation complexity of  $O(I)$ . Altogether this case will have the computation complexity of  $O(L)$ .

In summary, the nodes and resources selection step will have the computation complexity of  $O(L)$ .

Third, for the scheduling step, first the MMA algorithm will compute *MIN-AVG-EEAT-ETAT* for each ready task on each selected resource. Suppose the average number of capable resources for each ready task is  $R$ , this step will have the computation complexity of  $O(T*R)$ . Then, to find the task with the minimum *MIN-AVG-EEAT-ETAT* of all tasks, the EMM algorithm will have the computation complexity of  $O(T)$ . Considering the loop for scheduling all tasks, with a maximum of  $T*(T*R+T)$ , this step will have the computation complexity of  $O(T^2*R)$ .

In summary, with a maximum of  $T+L+T*(T*R+T)$ , the MMA algorithm will have the computation complexity of  $O(L+T^2*R)$ , where  $L$  is the length of the hash table on the monitor node,  $T$  is the number of all tasks, and  $R$  is the average number of capable resource for each task.

## 4.5 Example for Demonstration

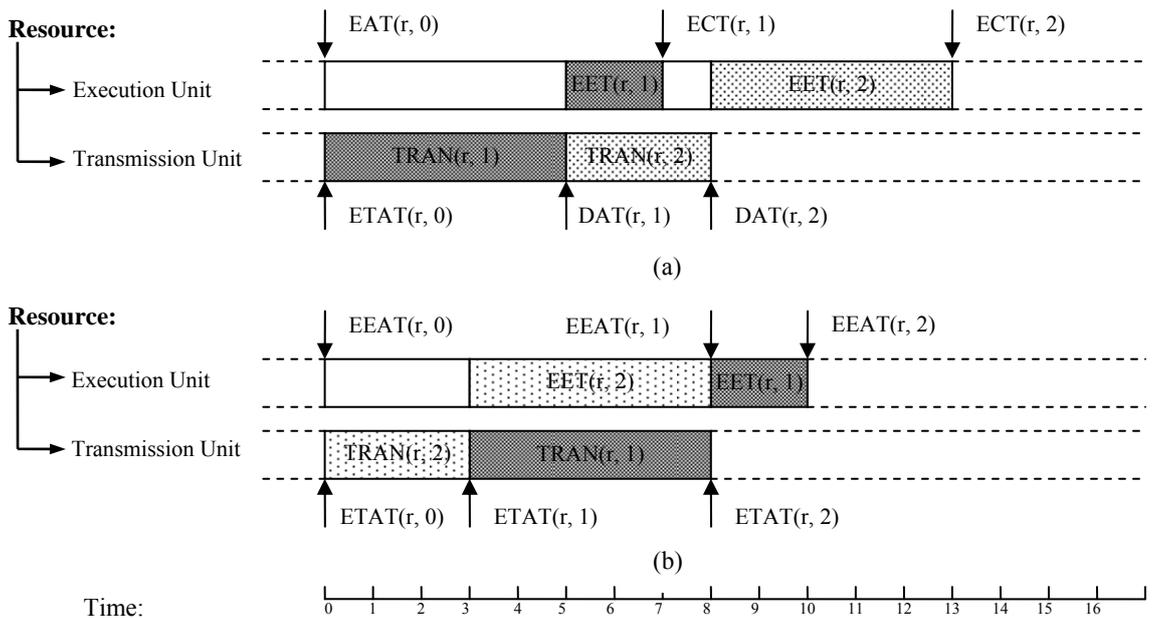
The Min-Min algorithm is a popular practical scheduling algorithm which is used in Kepler [LABH+2006] and vGrADS [BJDG+2005]. It can be described briefly as follows:

- (1) For each available task, find the resource  $r$  with the minimum  $ECT(r, t)$  value Find the minimum  $ECT(r, t)$  value over all available tasks

(2) Schedule the task with the minimum  $ECT(r, t)$  value to the resource that gives this value.

(3) Repeat this procedure until all the tasks have been scheduled.

Using the Min-Min and MMA algorithms respectively, we consider an example shown in Figure 4.4. In this example, resource  $r$  has both  $EEAT(r, 0) = OEAT(r) = 0$  and  $ETAT(r, 0) = OEAT(r) = 0$ , where  $EEAT(r, 0)$  is the time at which  $r$  will become free to perform any task and  $ETAT(r, 0)$  is the time at which the data transfer unit of  $r$  will become free to transmit any data. There are 2 tasks:  $task1$  with  $DAT(r, 1) = 5$  and  $EET(r, 1) = 2$ ,  $task2$  with  $DAT(r, 2) = 3$  and  $EET(r, 2) = 5$  which are needed to be scheduled on  $r$ .



**Figure 4.4 Scheduling Results of Original Min-Min Algorithm (a) and MMA Algorithm (b)**

If we apply the Min-Min algorithm, we will get  $ECT(r, 1) = EET(r, 0) + \max(EAT(r, 1), DAT(r, 1)) = 2 + \max(0, 5) = 7$  and  $ECT(r, 2) = EET(r, 0) + \max(EAT(r, 2), DAT(r, 2)) = 5 + \max(0, 3) = 8$  so we will schedule  $task1$  first and  $task2$  next and the total completion time will be 13. The scheduling result is shown in Figure 4.4(a).

If we apply the MMA algorithm, the scheduling result will be different. As Figure 4(b) shows, if we schedule *task1* on resource *r* first, we will get  $AVG-EEAT-ETAT(r, 1) = (EEAT(r, 1) + ETAT(r, 1))/2 = (5 + 7)/2 = 6$ , and if we schedule *task2* first, we will get  $AVG-EEAT-ETAT(r, 2) = (EEAT(r, 2) + ETAT(r, 2))/2 = (3 + 8)/2 = 5.5$ . Because  $AVG-EEAT-ETAT(r, 2) < AVG-EEAT-ETAT(r, 1)$ , this algorithm will schedule *task2* first and *task1* next and the total completion time will be 10. As Figure 4.4(b) shown, the total execution time is shorter than that by Min-Min.

## 4.6 Comparison and Simulation

In this section, we introduce the criteria for comparison first, followed by an example to demonstrate the advantages of MMA over Min-Min when it comes to scheduling instance-intensive workflows with considerable communication overheads involved.

Finally we will perform an experimental simulation in the Swinburne Workflow Test Environment. Our aim is to build the fundamental infrastructure depicted in Section 4.3 (denoted as SwinDeW-Cs: **S**winburne **D**ecentralised **W**orkflow for **C**loud based on **S**ervice) on top of it and perform the original Min-Min algorithm and MMA algorithm on SwinDeW-Cs in order to compare their performance.

Due to the reason described in Section 3.4.2, we only compare our algorithm with the Min-Min algorithm. Our objective is to perform both the Min-Min algorithm and the MMA algorithm on SwinDeW-Cs in order to compare the scheduling performance of both algorithms.

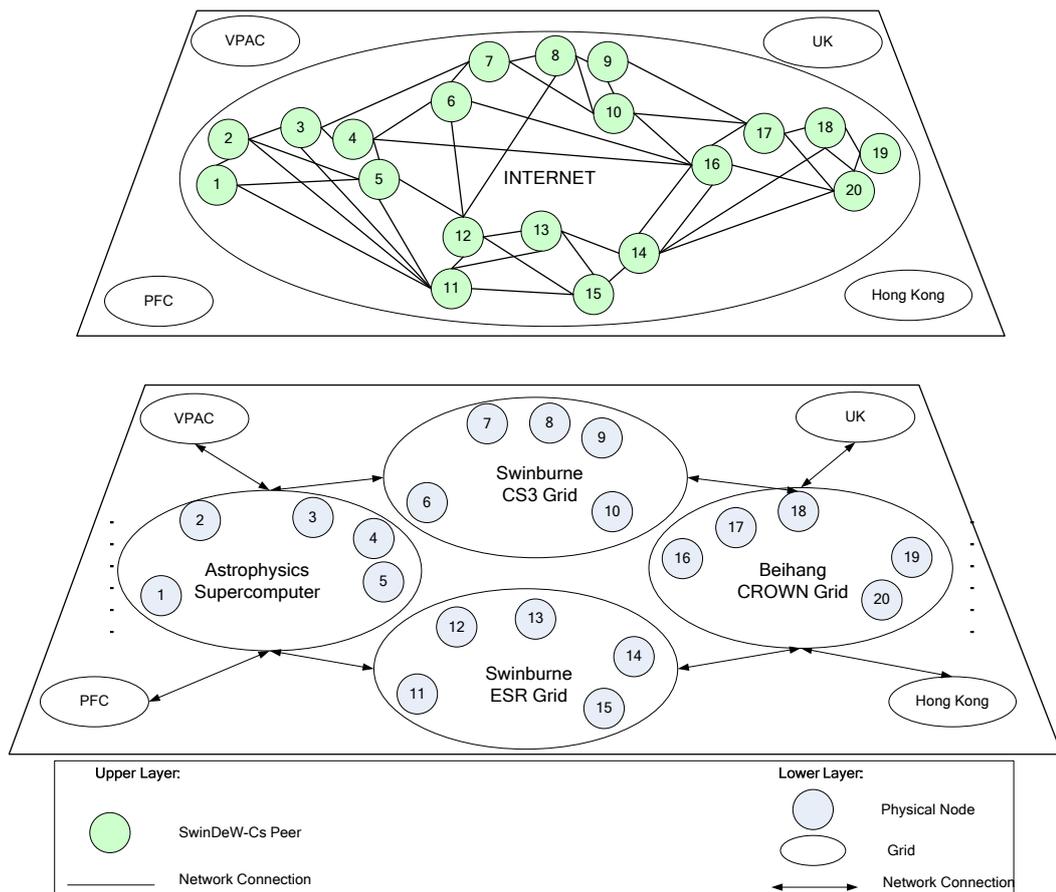
### 4.6.1 Simulation Environment

As shown in Figure 4.8, the SwinDeW-Cs (**S**winburne **D**ecentralised **W**orkflow for **C**loud based on **S**ervices) is also built on top of the Swinburne Workflow Test Environment (SWTE). The upper layer represents the logic infrastructure of scheduling algorithm, while the lower layer represents the SWTE. The numbers in the figure represents the mapping relationship of logic scheduling peers and physical nodes, i.e., if the number on a logic scheduling peer on upper layer is the same with that of a physical

node on lower layer, it implies that this logic scheduling peer is implemented on this physical node.

To simulate the services scattered on the Internet, it can be seen from Figure 4.5 that the concept of physical grids on the lower layer no longer exists on the upper layer. On the contrary, each node is implemented as an independent server, no matter on which grid it is located. As described previously in Section 3.6.1, the nodes are located all around the world, thus the SwinDeW-Cs environment can simulate the Internet environment effectively.

The architecture of SwinDeW-Cs is shown in Figure 4.5 with SWTE listed at the bottom for reference.



**Figure 4.5 Swinburne Decentralised Workflow for Cloud based on Service (SwinDeW-Cs)**

## 4.6.2 Criteria for Comparison

Similar to Section 3.6.2, the main criteria for workflow scheduling algorithms include mean execution time, resource utilisation rate and the overall throughput, where the overall throughput is the ultimate goal in instance-intensive workflows.

- *Mean execution time* is the time spent from the beginning of the first task to the end of the last task for a workflow instance. In a batch scheduling scenario like Instance-Intensive cloud workflows, the mean execution time is more suitable for assessing the scheduling performance. The shorter the mean execution time is, the better the performance is.
- *Resource utilisation rate* is the percentage of time a resource is busy. Low utilisation means a resource is idle and wasted. For scheduling Instance-Intensive cloud workflows involving considerable communication overheads, it includes two aspects: the execution unit utilisation rate and the data transfer unit utilisation rate. The higher the utilisation rate is, the better the performance is.
- *Throughput* is the ability of a resource to process a certain number of instances in a given period of time. It is also an important criterion in workflow scheduling, especially in Instance-Intensive cloud workflows. The higher the throughput is, the better the performance is.

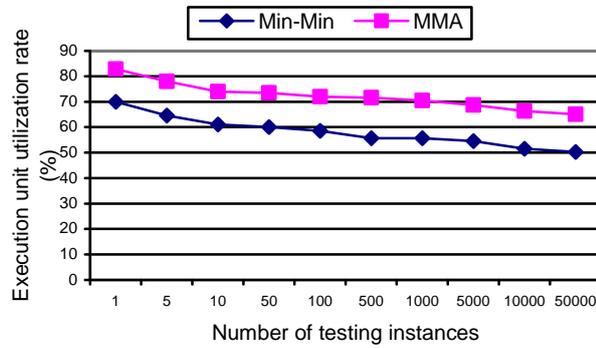
## 4.6.3 Simulation Process

Again as mentioned in Section 1.1, instance-intensive cloud workflows are *workflows with a huge number of (hence instance-intensive) concurrent workflow instances, usually much simpler than those complex scientific workflows, enabled on a cloud computing environment (hence cloud workflows)*. To simulate such workflows on service-based cloud computing environment, we construct the simulation environment focusing on the three aspects. As the first and second aspects are identical to those already illustrated in Section 3.6.3, we only list the new aspect. For the MMA algorithm, we use testing nodes located in different places connected via the Internet, thus the communication overheads vary.

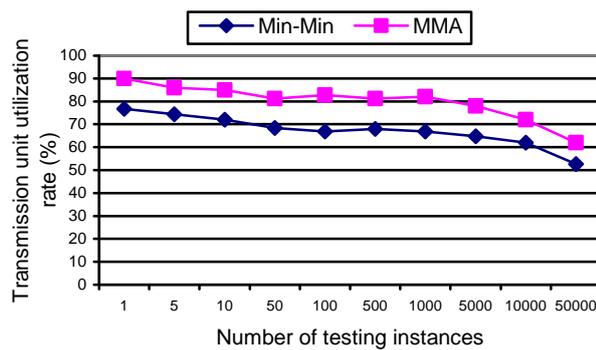
In the simulation, we first implement the original Min-Min algorithm on SwinDeW-Cs, then implement the Min-Min-Average algorithm, and compare them based on the three criteria mentioned earlier respectively.

#### 4.6.4 Results and Analysis

In Figure 4.6(a) and Figure 4.6(b), the horizontal axes represent the number of instances we put into the simulation, and the vertical axes represent the execution unit utilisation rate and transmission unit utilisation rate respectively. We can see that the execution unit utilisation rate and transmission unit utilisation rate of the MMA algorithm are about 20% higher than those of the original Min-Min algorithm.



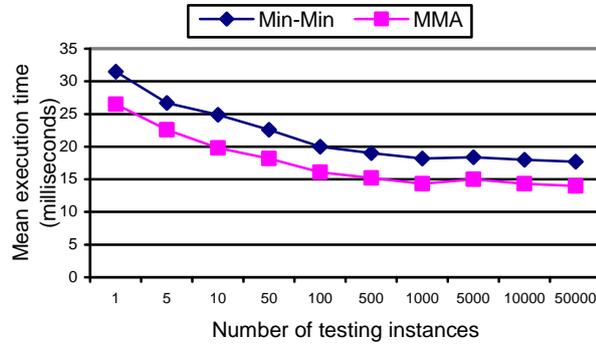
**Figure 4.6(a): Execution Unit Utilisation Rate of Min-Min and MMA Algorithms**



**Figure 4.6(b): Transmission Unit Utilisation Rate of Min-Min and MMA Algorithms**

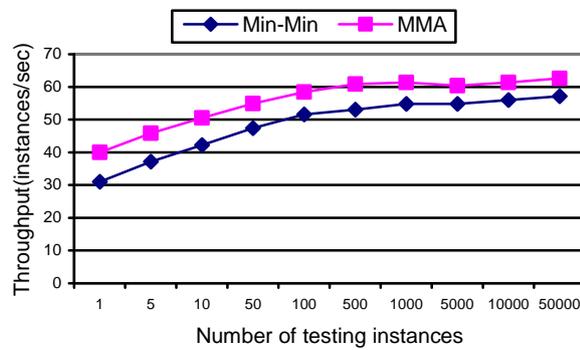
In Figure 4.7, the horizontal axis is the number of instances we put into the simulation, and the vertical axis is the mean execution time of all instances. It can be

seen that the mean execution time of the MMA algorithm is about 20% shorter than that of the original Min-Min algorithm.



**Figure 4.7: Mean Execution Time of Min-Min and MMA Algorithms**

In Figure 4.8, the horizontal axis represents the number of instances we put into the simulation, and the vertical axis represents the overall throughput. We can see that the overall throughput of the MMA algorithm is about 30% higher than that of the Min-Min algorithm.



**Figure 4.8: Overall Throughput of Min-Min and MMA Algorithms**

In conclusion, the MMA algorithm has a higher resource utilisation rate - which includes execution unit and transmission unit utilisation rates - than the Min-Min algorithm, and thus has a shorter mean execution time and a higher overall throughput than Min-Min algorithm when scheduling instance-intensive cloud workflows with considerable communication overheads involved.

## 4.7 Summary

As mentioned in Section 1.2, the most common infrastructures of cloud computing environment consist of physically collocated grids (grid-based) and of services scattered on the Internet (service-based). This chapter mainly focus on workflow scheduling on a service-based cloud computing platform, while scheduling instance-intensive workflows on grid-based cloud computing platform has already been discussed in Chapter 3 and scheduling instance-intensive workflows on business-based cloud computing platform will be discussed in Chapter 5.

In this chapter, we have proposed a new algorithm called the MMA (Min-Min-Average) algorithm for efficiently scheduling instance-intensive cloud workflows involving considerable communication overheads. This algorithm is based on the popular Min-Min algorithm but takes the unique characteristics of instance-intensive workflows involving considerable communication overheads into consideration. There are two major contributions in this chapter: one is the novel design which can make the scheduling algorithm adapt to the change of network transmission speed dynamically and the other is the innovative scheduling strategy which can increase the overall throughput significantly. The simulation has demonstrated that our MMA algorithm is more efficient when scheduling such workflows.

## Chapter 5

# Compromised-Time-Cost Algorithm for Cost-constrained Cloud Workflows

This chapter is based on our paper [YLCL+2008] and is organised as follows. In Section 5.1, we give a brief introduction to the background of the proposed algorithm, followed by the requirements analysis in Section 5.2. Section 5.3 presents fundamental design for our cloud workflow system: SwinDeW-Cb, Section 5.4 discusses the details of the scheduling algorithm, and Section 5.5 gives an example for illustration. Later Section 5.6 demonstrates the benefits of our work via simulation and comparison. Finally in Section 5.7, we summarise this chapter.

### 5.1 Introduction

With the promotion of world's leading companies, cloud computing is attracting more and more attention of researchers recently. As defined in Section 2.1.2.1, cloud computing has many potential advantages which include *lower cost, device and location independence* and so on [VRCL2008]. These advantages can also benefit workflow systems built on top of it.

In commercial cloud computing environment, users pay for what they use each time (pay per use). We define this cloud computing environment as business-based cloud computing environment. Cost-constraint instance-intensive workflows can be executed on this cloud computing environment for a reasonable price to save the expenditure of purchasing and maintaining resources and (software) services. In this case, they are

characterised as a huge number of workflow instances (hence instance intensive) and are bounded by a certain budget for execution (hence cost constrained) on a cloud computing platform (hence cloud workflows).

This chapter mainly focuses on scheduling algorithms for instance-intensive cost-constrained workflows on business-based cloud computing environment, while scheduling algorithms for instance-intensive cloud workflows on grid-based cloud computing platform and service-based cloud computing platform have already been discussed in Chapter 3 and Chapter 4 respectively.

## **5.2 Requirements Analysis and Overall Solution**

### **5.2.1 Requirements Analysis**

On one hand, users are always concerned about the execution time of workflows. On the other hand, on a cloud computing platform, users do not need to have any special resources or services of themselves to run their applications. In reality, some cloud services provided, for example, by Google or Amazon, are free. However, cloud services for execution of sophisticated scientific and business workflow applications are unlikely to be free. This means that users may need to pay for use. Because users are normally sensitive to execution cost, it is another major concern for cloud workflow applications. Given that the attributes of both time and cost are involved, it is necessary to enable users to compromise for better user satisfaction. In particular, during workflow execution, in some circumstances, users may decide on the fly to pay slightly more to reduce execution time, or save execution cost by allowing longer execution time as long as the final deadline can be met.

Moreover, workflow scheduling designers should consider the characteristics of cloud computing when designing cloud workflow scheduling algorithms. Let us have a look at the essence of cloud computing first. Generally speaking, cloud computing has the following major characteristics [BMQL+2008]: I) it hosts a variety of different loads, including batch-style back-end jobs and interactive, user-facing applications, which means the load of servers in cloud is highly dynamic; II) it allows loads to be deployed and scaled-out quickly through the rapid provisioning of virtual machines or physical

machines, which means it is highly scalable; III) it supports redundant, self-recovering, highly scalable programming models that allow loads to recover from many unavoidable hardware/software failures; and IV) it rebalances the allocations when needed in real time. All these characteristics should be considered in the cloud workflow scheduling algorithm design.

From the analysis above, we can summarise the primary requirements for designing scheduling algorithms for instance-intensive cost-constrained cloud workflows.

- First, the scheduling algorithms should take execution cost of pay for use as a key factor.
- Second, as an important criterion of instance-intensive workflows, mean execution time will be taken as another key factor.
- Third, the algorithms should facilitate multiple strategies for compromising execution time and cost with user intervention enabled on the fly.
- Finally, they should conform to the nature of cloud computing, particularly, the scheduling algorithms should consider the background load of the servers when calculating the task execution time. After tasks are distributed to a server, actions should be taken when scheduled tasks are not completed successfully due to hardware or software failures.

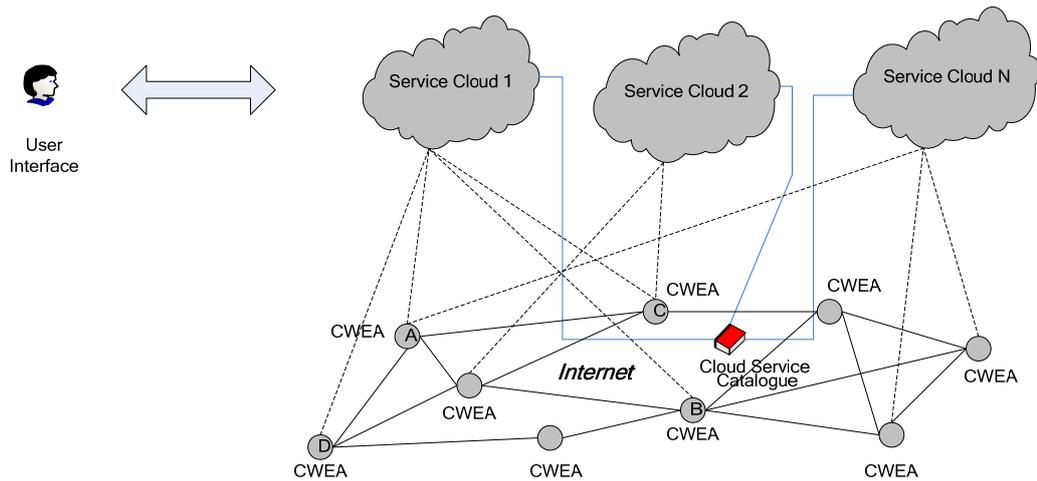
### **5.2.2 Overall Solution**

In order to meet the requirements mentioned above, we propose a Compromised-Time-Cost algorithm (CTC) for cost-constrained cloud workflows. In CTC algorithm, we compromise the time and cost throughout the scheduling process. This algorithm can be further divided into two sub-algorithms. The CTC-MC (Compromised-Time-Cost algorithm Minimising execution Cost) algorithm minimises the execution cost within user designated deadline, and the CTC-MT (Compromised-Time-Cost algorithm Minimising execution Time) algorithm minimises the execution time within user

designated budget. However, both algorithms can enable user to input an acceptable compromise between execution time and execution cost on the fly.

### 5.3 Scheduling Infrastructure

As depicted in Figure 5.1, our SwinDeW-Cb (**S**winburne **D**ecentralised **W**orkflow for **C**loud based on **B**usiness model) cloud workflow architecture is divided into the following major parts at a high level: dynamic service clouds, cloud workflow execution agents (CWEA), a cloud service catalogue, and a user interface. The other general parts of a cloud computing platform such as system management, provisioning tool, monitoring and metering are not shown explicitly in the figure because we focus on the discussion of cloud workflow systems.



**Figure 5.1: Architecture of SwinDeW-Cb**

#### 5.3.1 Service Cloud

The servers with the same service are organised dynamically as a service cloud. It should be noted that all service clouds are independent of each other and every server will automatically join the service clouds according to the services it can provide. For example, server *A* can provide service *1* and service *N*, thus it joins Service Cloud *1* and Service Cloud *N*. Server *C* can provide service *1* and service *2*, thus it joins Service Cloud *1* and Service Cloud *2* correspondingly.

In a service cloud, every server tracks the information of its neighbours so the scheduling is always performed cooperatively with its neighbours. For example, server *A* tracks the information of server *C* and server *D* for they all join the same Service Cloud. Unlike in SwinDeW-Cs, where ordinary nodes track nearest neighbours and monitor node tracks most diverse neighbours, the nodes in SwinDeW-Cb track neighbours of most diverse service providers.

### **5.3.2 Cloud Workflow Execution Agent (CWEA)**

To efficiently manage the services, each service in service clouds is administrated by a cloud workflow execution agent (CWEA). The CWEA manages the services during workflow scheduling and execution, including service monitoring, cooperative scheduling and execution with other agents which manage the services of the same type.

The task allocation among service clouds can be described as follow. After a task which requests a specific service for execution has been scheduled to a service cloud, a suitable node in the service cloud will be selected for real execution according to the applied strategy, such as achieving load-balance, minimum cost or shortest distance to the task host etc.

### **5.3.3 Cloud Service Catalogue**

Cloud services are a foundation of cloud computing. There are a variety of services available over the Internet that deliver computing functionality on the service provider's infrastructure. To access the services, there is a Cloud Service Catalogue which is a list of services that a user can request. In workflow systems, a task usually needs a cloud service to execute. This cloud service is registered along with other cloud services by service providers on the Cloud Service Catalogue Server. Usually, the items in the catalogue are maintained by their corresponding service clouds via the heartbeat mechanism.

### **5.3.4 User Interface**

SwinDeW-Cb provides a user interface for users to monitor the workflow execution status and more importantly, to provide input on the setting of compromised time and

cost on the fly if needed. The input from the user on time and cost will be taken into account for scheduling the next round tasks in order to achieve better user satisfaction.

## 5.4 CTC Scheduling Algorithm

Considering the nature of cloud computing and the application scenario of instance-intensive cost-constrained workflows, we propose a compromised-time-cost (CTC) scheduling algorithm which focuses on the following perspectives against the requirements.

- First, considering the pay for use feature of cloud workflows, the CTC algorithm takes execution cost and execution time as the two key considerations. The CTC algorithm can either minimise the cost under certain user-designated deadlines (denoted as CTC-MC) or minimise the execution time under certain user-designated cost (denoted as CTC-MT).
- Second, different from other scheduling algorithms, our CTC algorithm always enables the compromise of execution cost and time. The algorithm provides a just-in-time graph of time-cost relationship during workflow execution in the user interface for users to choose an acceptable compromise before the next round scheduling begins if they wish. If no user input is detected at the time of the next scheduling round, the default scheduling strategy will be automatically applied so no delay will be caused. More details are in Step 4 of the algorithm described later.
- Third, the algorithm considers sharing, conflicting and competition of services caused by concurrent multiple instances running on the highly dynamic cloud computing platform. For example, instead of services that are normally available when only considering a single workflow instance, lower cost services may be heavily competed and hence unavailable at the time whilst waiting for their availability could cause significant and unnecessary delay.

Therefore, our algorithm considers the following aspects: 1) background load: in order to estimate the execution time more accurately, background load is considered

when calculating the task execution time on each specific server; 2) dynamic adjustment: in order to adapt to the dynamic change of load, after tasks are distributed to a server, the server may reschedule the tasks when encountering heavy load; 3) checking and rescheduling: in order to deal with the execution failures, uncompleted tasks will be rescheduled with a high priority for the next round.

Accordingly, our scheduling algorithm has a pre-step to discover and reschedule the tasks which failed to complete due to various reasons on the cloud computing platform or distribute tasks to other light-load servers when the host server encountering heavy load. The detailed descriptions of the CTC-MC and CTC-MT algorithm are listed in Section 5.4.1 and Section 5.4.2 respectively.

#### 5.4.1 CTC-MC Scheduling Algorithm

Before we give the details of the steps, in order to have an overview, we summarise the CTC-MC algorithm in Table 5.1 and present the algorithm details next.

**Table 5.1: Overview of CTC-MC Algorithm**

<b>Algorithm: CTC-MC Scheduling</b>	
	<b>Input:</b> An array of workflow instances <b>Output:</b> A schedule (lowest cost by default)
Pre-Step	0.1 Check uncompleted tasks and schedule them first in this round.
Step 1	1.1 Calculate the sub-deadlines for tasks of the last instance. 1.2 Calculate the sub-deadlines for tasks of other instances based on the last instance, assuming that the schedule follows the stream-pipe mode to avoid competitions for cheaper services.
Step 2	2.1 Calculate the estimated execution time and cost of each ready task on each available service.
Step 3	3.1 Allocate each task to the service which gives the execution time that does not exceed the sub-deadline of the task with the lowest cost.
Step 4	4.1 Provide just-in-time time-cost relationship graph for the user to optionally choose an updated compromised deadline for scheduling with lowest cost by default.
Step 5	5.1 Repeat the above steps for next round scheduling.

### **Pre-Step: Check uncompleted tasks and schedule them first in this round**

The main purpose of this step is to discover the unsuccessful tasks or redistribute tasks to other light-load servers when the host server encountering heavy load. For the former, each task has been allocated a timer once it is scheduled to other servers. If a timer of a certain task expires and the task has not been completed yet, the execution will be considered unsuccessful. For the latter, all tasks allocated to such host servers have to be rescheduled. In order to give a higher priority to these tasks, they will be scheduled first before new available tasks.

### **Step 1: Allocate sub-deadlines to tasks for each instance**

Similar to the overall deadline of the entire workflow instance, a sub-deadline is the latest completion time allocated to a single task, which the completion time of the task should not exceed. In Deadline-MDP, the sub-deadline is assigned by combining Breadth-First Search (BFS) and Depth-First Search (DFS) algorithms with critical path analysis to compute start times, proportion and sub-deadlines of every task partition, using the following policies [YBT2005]:

*P1. The cumulative sub-deadline of any independent path between two synchronization tasks must be same.*

*P2. The cumulative sub-deadline of any path from entry  $i$  to exit  $j$  is equal to the user-designated deadline.*

*P3. Any assigned sub-deadline must be greater than or equal to the minimum processing time of the corresponding task partition.*

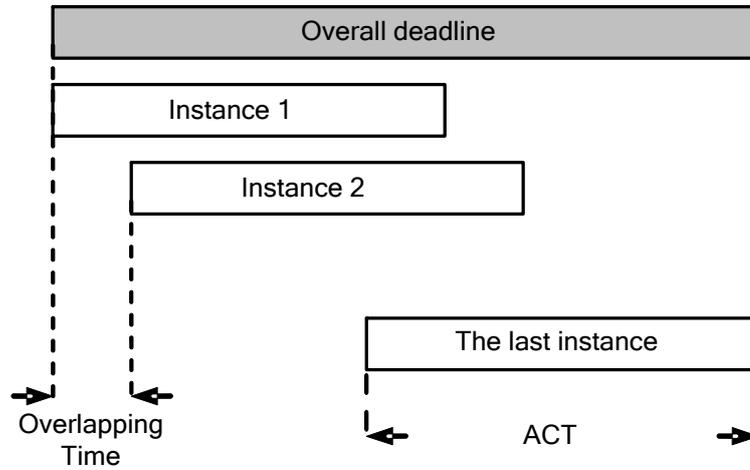
*P4. The overall deadline is divided over task partitions in proportion to their minimum processing time.*

In Deadline-MDP, the sub-deadlines of each instance are the same for they are all calculated from the same DAG. However, CTC-MC deals with multiple instances rather than a single instance in Deadline-MDP. We find that assigning separate sub-deadlines to each instance is more efficient. Thus the CTC-MC algorithm only calculates sub-deadlines of tasks of the last instance of the same type once and deducts sub-deadlines of others from that result.

In CTC-MC algorithm, we assume the schedule follows the stream-pipe mode. The ground for this assumption is to stagger the deadlines of the large number of concurrent instances of the same nature to avoid the fierce competition of cheaper services. We will see later in this step that if we set sub-deadlines of these instances the same, service utilisation of cheaper services will drop due to competition within a certain period of time. However, if we stagger the deadlines, the tasks with later sub-deadlines will have a chance to use cheaper services after they are released by tasks with earlier sub-deadlines. Of course, these workflow instances can still be executed in parallel as long as the completion time of each task does not exceed its sub-deadline allocated. With this assumption, from Figure 5.2, we have:

$$\mathbf{ACT + (n - 1) \times OT = OD} \quad (1)$$

where *ACT* = average completion time, *OT* = overlapping time, *OD* = overall deadline



**Figure 5.2: Stream-pipe Mode Sub-deadline Distribution**

After the overlapping time is calculated, we can calculate sub-deadlines for tasks of other instances based on the sub-deadline distribution of the last task. The sub-deadline of each task of the  $i^{th}$  instance equals to the sub-deadline of the corresponding task of the last instance minus  $(n-i) \times \text{overlapping time}$  where  $n$  is the total number of workflow instances, except the last task whose sub-deadline equals the overall deadline.

**Step 2: Calculate estimated execution time and cost on each service**

First, let us list the price tables. As for Table 5.2, it can be seen that higher processing speed, which can reduce the execution time, usually ends up with higher cost. As for Table 5.3, higher bandwidth which can reduce transmission time normally results in higher cost. The round trip time is not listed here explicitly, but is considered when calculating transmission time.

**Table 5.2. Service Processing Speeds and Corresponding Prices for Task Execution**

Service ID	Processing speed	Cost (\$/Task)
1	0.5	3
2	1	6
3	2	12
4	4	24

**Table 5.3. Service Bandwidths and Corresponding Prices for Data Transmission**

Service ID	Bandwidth (Mbps)	Cost (\$/MB)
1	100	1
2	1000	10
3	10000	100
4	100000	1000

1) For each service ID, suppose the processing speed is PS and the current load is L, we can calculate the execution time and cost for executing a task on this service as follows.

$$ET = OET/PS/(1 - L) \quad (2)$$

where  $ET$  = execution time,  $OET^1$  = original execution time,  $PS$  = processing speed,  $L$  = load

$$EC = \text{cost}(PS) \quad (3)$$

where  $EC$  = execution cost,  $\text{cost}(PS)$  is a function whose value can be looked up in Table 5.2

2) Suppose the round trip time from the service on which the previous task is executed is RTT, for each service ID, we calculate the transmission time and cost as follows.

---

<sup>1</sup> OET is the execution time of the task on a standard reference service.

$$\mathbf{TT = RTT + DS/B} \quad (4)$$

where  $TT$  = transmission time,  $DS$  = data size,  $B$  = bandwidth

$$\mathbf{TC = cost(B)} \quad (5)$$

where  $TC$  = transmission cost,  $cost(B)$  is a function whose value can be looked up in Table 5.3

3) Then the total execution time and cost can be calculated as follows in formulas (6) and (7) respectively:

$$\mathbf{FCT = \max(SAT, DAT) + ET} \quad (6)$$

where  $FCT$  = final completion time,  $SAT$  = service available time,  $DAT$  = data available time, which is the time at which all data needed for task execution arrived.

$$\mathbf{FTC = EC + TC} \quad (7)$$

where  $FTC$  = final total cost

### **Step 3: Allocate tasks to services**

$FCT$  and  $FTC$  are calculated one task at a time, following the rule of FCFS (First Come First Served). The CTC algorithm allocates each task to the corresponding service which gives the execution time that does not exceed the sub-deadline at the lowest total cost. After a task is scheduled to a service, the new service available time of the service is calculated as follow:

$$\mathbf{new SAT = prev SAT + ET} \quad (8)$$

where  $SAT$  = service available time

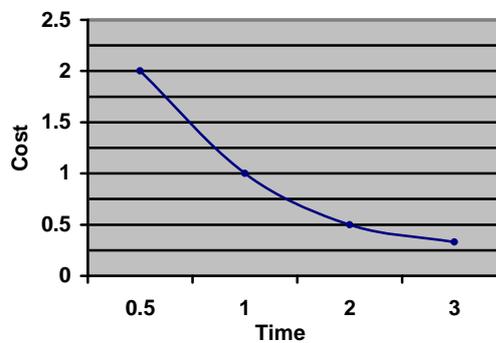
It should be addressed that after the service available time has been updated, all  $FCT$  and  $FTC$  of the remaining tasks on this service are recalculated based on the update. This procedure repeats until all remaining tasks are scheduled.

### **Step 4: Provide just-in-time time-cost relationship graph for user to optionally choose an updated compromised deadline for scheduling**

For user's convenience, the CTC algorithm provides a time-cost relationship graph on the fly for user to decide the scheduling objective, at the time of each scheduling round

optionally. This graph is drawn by getting the total execution cost under the assumed overall deadlines of 0.25, 0.5, 2 and 4 times of the original user-designated deadline, applying the previous steps.

During this on-demand interaction, a user can decide whether to reduce execution time by paying slightly more or slightly postponing the deadline to reduce execution cost. An example of such a graph is shown in Figure 5.3. The user has the opportunity to redefine the deadline or cost in each scheduling round on the fly to achieve more satisfactory compromised-time-cost outcomes.



**Figure 5.3 Example of Time-Cost Relationship Graph**

It should be noted that the algorithm currently described is the strategy to achieve the low execution cost when the user sets the deadline on the fly. For simplicity without losing generality, we do not address the other strategy of the algorithm that is similar to the procedure above with cost and time swapped and can achieve the faster execution time when the user sets the execution cost on the fly. The provision of two strategies give user maximum flexibility and hence more satisfaction.

#### **Step 5: Next round scheduling**

After the user has chosen an updated compromised deadline, the CTC-MC algorithm will repeat all the steps to find a schedule conforming to the objective based on the new deadline for the next round. If no user input is provided, lowest execution cost, the default scheduling strategy, will be applied.

### 5.4.2 CTC-MT Scheduling Algorithm

The goal of CTC-MT algorithm is to minimise execution cost under user-designated budget  $UB$ . It is actually based on CTC-MC algorithm and uses a Binary Search strategy. It is described as follow:

- First, because the maximum overall deadline ( $maxEOD$ ) is equal to the total execution time if all tasks are scheduled on the cheapest and slowest services, CTC-MT sets *lower bound* to 0, *upper bound* to  $maxEOD$ , and begins the Binary Search.
- *Current overall deadline* is set to  $(lower\ bound + upper\ bound)/2$ , and the *cost* within *current overall deadline* is calculated based on CTC-MC algorithm.
- If the cost is less than  $UB$ , it means that it is possible to execute some tasks on faster but more expensive services to shorten the total execution cost. The new *upper bound* is set to half of its current value.
- If the cost is greater than  $UB$ , it means that it is impossible to complete tasks within this budget because CTC-MC always seeks the minimum cost at each step. We have to increase the *current overall deadline* to allow more tasks to be executed on slower but cheaper services to reduce the cost so that it does not exceed  $UB$ . The new *lower bound* is set to *current overall deadline*.
- The iteration repeats until the cost is equal to  $UB$ , which is perfect but unlikely in most circumstances, or is not changed again, where in this circumstance, the cost should be slightly less than  $UB$ .
- Provide a just-in-time time-cost relationship graph for the user to optionally choose an updated compromised deadline for scheduling. The (deadline, cost) value pairs in the graph are tracked from the previous iterations.

- After the user has chosen an updated compromised deadline, the schedule within this designated deadline and cost is performed. If no user input is provided, the schedule within the calculated *current overall deadline*, will be applied.

### 5.4.3 Complexity Analysis

#### 5.4.3.1 Computation Complexity of CTC-MC Algorithm

For the CTC-MC algorithm, for pre-step, it checks uncompleted tasks and schedules them first in this round. Because the average number of uncompleted tasks is usually much smaller than tasks that will be scheduled normally at next steps, the computation complexity of this step will be covered by that of next steps.

For the first step, the CTC-MC algorithm calculates the sub-deadlines for tasks of the last instance and the sub-deadlines for tasks of other instances based on the last instance. It will have the computation complexity of  $O(T)$ , where  $T$  is the number of all tasks.

For the next step, the CTC-MC algorithm calculates the estimated execution time and cost of each ready task on each available service. Suppose the average number of capable resources for each task is  $R$ , this step will have the computation complexity of  $O(T*R)$ . Then, the CTC-MC algorithm allocates each task to the service which gives the execution time that does not exceed the sub-deadline of the task with the lowest cost. This step will also have the computation complexity of  $O(T*R)$ .

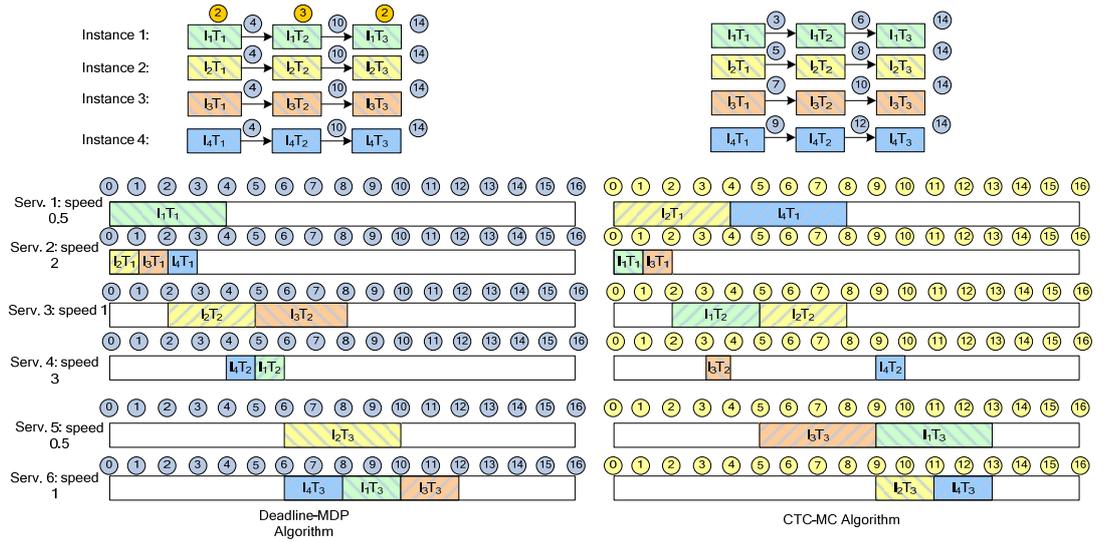
Considering the loop for scheduling all tasks, with a maximum of  $T+T*(T*R+T*R)$ , the CTC-MC algorithm have the computation complexity of  $O(T^2*R)$ , where  $T$  is the number of all tasks, and  $R$  is the average number of capable resources for each task.

#### 5.4.3.2 Computation Complexity of CTC-MT Algorithm

For the CTC-MT algorithm, it actually calls the CTC-MC algorithm using a binary search strategy. The CTC-MT algorithm have the computation complexity of  $O(I*T^2*R)$ , where  $I$  is the maximum iteration number defined,  $T$  is the number of all tasks, and  $R$  is the average number of capable resources for each task.

## 5.5 Example for Demonstration

For convenience, we use  $I_i T_j$  to represent the  $j^{\text{th}}$  task of the  $i^{\text{th}}$  instance. Here is an example for illustration and the conditions are listed below:



**Figure 5.4 Comparison of Deadline Distributions**

- There are 4 workflow instances from the same workflow model that need to be scheduled, and the user gives a total deadline of 14 seconds. The DAG of the workflow model is shown on the top of Figure 5.4.
- The estimated execution times of task  $T_1$ ,  $T_2$  and  $T_3$  are 2, 3, and 2 seconds respectively, and the minimum execution time follows this ratio.
- The estimated execution costs of task  $T_1$ ,  $T_2$  and  $T_3$  are 20, 30, and 40 dollars respectively, and the minimum execution cost follows this ratio.
- Task  $I$  can be executed on service  $I$  with an execution speed half of the estimated execution speed and an execution cost half of the estimated execution cost, or executed on service 2 with an execution speed of twice the average execution speed and an execution cost of twice the estimated execution cost.

- Task 2 can be executed on service 3 with an execution speed of exactly the estimated execution speed and an execution cost of exactly the estimated execution cost, or on service 4 with an execution speed triple the estimated execution speed and an execution cost triple the estimated execution cost.
- Task 3 can be executed on service 5 with an execution speed of half the estimated execution speed and an execution cost of half the estimated execution cost, or on service 6 with an execution speed exactly the estimated execution speed and an execution cost exactly the estimated execution cost.
- The communication time of every two tasks executed on every two services is exactly 1 second.

### 5.5.1 Scheduling Procedure of Deadline-MDP algorithm

#### Deadline Distribution

According to  $P4$ , the overall deadline is divided over task partitions in proportion to their minimum processing time. Giving the overall deadline of 16, after applying this policy to Instance 1, we will get  $sub\_deadline(I_1T_1) = 4$ ,  $sub\_deadline(I_1T_2) = 10$  and  $sub\_deadline(I_1T_3) = 14$ . As the Deadline-MDP does not stagger the deadline of instances, the sub-deadlines of the corresponding tasks of other instances are equal to the calculated sub-deadlines.

#### Schedule Round 1

In the first round, start tasks such as  $I_1T_1$ ,  $I_2T_1$ ,  $I_3T_1$  and  $I_4T_1$  can be scheduled first. As both service 1 (denoted as  $S_1$ ) and service 2 (denoted as  $S_2$ ) are idle,  $service\_available\_time(S_1) = service\_available\_time(S_2) = 0$ , which means they are available from the beginning. The tasks are scheduled sequentially by the incoming sequence:  $I_1T_1$ ,  $I_2T_1$ ,  $I_3T_1$  and  $I_4T_1$

(1) For task  $I_1T_1$  which has a  $sub\_deadline = 4$ , if it is scheduled on  $S_1$  that has half normal speed and half normal cost, we will get the following outcomes:

- As  $I_1T_1$  is the start task, there is no data generated from predecessor tasks for it. Thus its data available time on  $S_1$  (denoted as  $data\_available\_time(I_1T_1|S_1)$ ) is 0.
- $final\_completion\_time(I_1T_1|S_1) = \max(service\_available\_time(S_1), data\_available\_time(I_1T_1|S_1)) + 2 * estimated\_execution\_time(I_1T_1|S_1) = \max(0, 0) + 2 * 2 = 4$  **second**
- $final\_execution\_cost(I_1T_1|S_1) = estimated\_final\_execution\_cost(I_1T_1) / 2 = 20 / 2 = 10$  **dollars**
- The new  $service\_available\_time(S_1) = final\_completion\_time(I_1T_1|S_1) = 4$  **second**

If it is scheduled on  $S_2$  that has twice normal speed and twice normal cost, we will get the following outcomes:

- $data\_available\_time(I_1T_1|S_2) = 0.$
- $final\_completion\_time(I_1T_1|S_2) = \max(service\_available\_time(S_2), data\_available\_time(I_1T_1|S_2)) + estimated\_execution\_time(I_1T_1|S_2) / 2 = \max(0, 0) + 2 / 2 = 1$  **second**
- $final\_execution\_cost(I_1T_1|S_2) = estimated\_final\_execution\_cost(I_1T_1) * 2 = 20 * 2 = 40$  **dollars**
- The new  $service\_available\_time(S_2) = final\_completion\_time(I_1T_1|S_2) = 1$  **second**

As both completion times do not exceed *sub-deadline* ( $I_1T_1$ ) and the execution cost on  $S_1$  is less than that on  $S_2$ ,  $I_1T_1$  is scheduled on  $S_1$  finally.

(2) For task  $I_2T_1$  which also has a sub-deadline = 4, if it is scheduled on  $S_1$  that only has a half normal speed and half normal cost, we will get the following outcomes:

- $data\_available\_time(I_2T_1|S_1) = 0.$
- $final\_completion\_time(I_2T_1|S_1) = \max(service\_available\_time(S_1), data\_available\_time(I_2T_1|S_1)) + 2 * estimated\_execution\_time(I_2T_1|S_1) = \max(4, 0) + 2 * 2 = 8$  **second**
- $final\_execution\_cost(I_2T_1|S_1) = estimated\_final\_execution\_cost(I_2T_1) / 2 = 20 / 2 = 10$  **dollars**

- The new *service\_available\_time* ( $S_1$ ) = *final\_completion\_time* ( $I_2T_1|S_1$ ) = 8 **second**

If it is scheduled on  $S_2$  which only has twice normal speed and twice normal cost, we will get the following outcomes:

- *data\_available\_time*( $I_2T_1|S_2$ ) = 0.
- *final\_completion\_time* ( $I_2T_1|S_2$ ) =  $\max$  (*service\_available\_time* ( $S_1$ ), *data\_available\_time*( $I_1T_1|S_1$ )) + 2\* *estimated\_execution\_time* ( $I_1T_1|S_1$ ) =  $\max$  (0, 0) + 2 \* 2 = 1 **second**
- *final\_execution\_cost*( $I_2T_1|S_2$ ) = *estimated\_final\_execution\_cost* ( $I_2T_1$ ) \* 2 = 20 \* 2 = 40 **dollars**
- The new *service\_available\_time* ( $S_2$ ) = *final\_completion\_time* ( $I_2T_1|S_2$ ) = 1 **second**

We can see that although the execution cost on  $S_1$  is less, the completion time on  $S_1$  will exceed the sub-deadline. Thus it has to be scheduled to the more expensive  $S_2$ .

(3) Following the same routine,  $I_3T_1$  and  $I_4T_1$  are also scheduled on  $S_2$ , the scheduling result is shown in Figure 5.3.

## Schedule Round 2

In the second round, tasks  $I_1T_2$ ,  $I_2T_2$ ,  $I_3T_2$  and  $I_4T_2$  can be scheduled because their predecessors have already been scheduled. Because both service 3 (denoted as  $S_3$ ) and service 4 (denoted as  $S_4$ ) are idle, thus *service\_available\_time* ( $S_3$ ) = *service\_available\_time* ( $S_4$ ) = 0, which means they are available from the beginning. We can see from Figure 5.4 that these tasks are available in sequence of  $I_2T_2$ ,  $I_3T_2$ ,  $I_4T_2$  and  $I_1T_2$ , which is determined by the completion time of their respective predecessors. Therefore, the tasks are scheduled in this sequence.

(1) For task  $I_2T_2$  which has a *sub-deadline* = 10, if it is scheduled on  $S_3$  that has exactly the normal speed and exactly the normal cost, we will get the following outcomes:

- Because  $I_2T_2$  has a predecessor  $I_2T_1$ , so  $data\_available\_time(I_2T_2|S_3) = final\_completion\_time(I_2T_1|S_2) + data\_transmission\_time(S_2, S_3) = 1 + 1 = 2$ .
- $final\_completion\_time(I_2T_2|S_3) = \max(service\_available\_time(S_3), data\_available\_time(I_2T_2|S_3)) + estimated\_execution\_time(I_2T_2|S_3) = \max(2, 0) + 3 = 5$  **second**
- $final\_execution\_cost(I_2T_2|S_3) = estimated\_final\_execution\_cost(I_2T_2) = 30$  **dollars**
- The new  $service\_available\_time(S_3) = final\_completion\_time(I_2T_2|S_3) = 5$  **second**

If it is scheduled on  $S_4$  which has twice normal speed and twice normal cost, we will get the following outcomes:

- $data\_available\_time(I_2T_2|S_4) = final\_completion\_time(I_2T_1|S_2) + data\_transmission\_time(S_2, S_4) = 1 + 1 = 2$ .
- $final\_completion\_time(I_2T_2|S_4) = \max(service\_available\_time(S_4), data\_available\_time(I_2T_2|S_4)) + estimated\_execution\_time(I_2T_2) / 3 = \max(0, 2) + 3 / 3 = 3$  **second**
- $final\_execution\_cost(I_2T_2|S_4) = estimated\_final\_execution\_cost(I_2T_2) * 3 = 30 * 3 = 90$  **dollars**
- The new  $service\_available\_time(S_4) = final\_completion\_time(I_2T_2|S_4) = 3$  **second**

As both completion times do not exceed *sub-deadline* ( $I_2T_2$ ) and the execution cost on  $S_3$  is less than that on  $S_4$ ,  $I_2T_2$  is scheduled on  $S_3$  finally.

(2) Following the same routine,  $I_3T_2$  and is scheduled on  $S_3$ ,  $I_4T_2$  and  $I_1T_2$  are also scheduled on  $S_4$ , the scheduling result is shown in Figure 5.4.

### Schedule Round 3

The tasks are available in sequence of  $I_2T_3$ ,  $I_4T_3$ ,  $I_1T_3$  and  $I_3T_3$ . Finally  $I_2T_3$  is scheduled on  $S_5$ ,  $I_4T_3$ ,  $I_1T_3$  and  $I_3T_3$  are scheduled on  $S_6$ , the scheduling result is shown in Figure 5.4.

### Execution Cost

The total cost of Deadline-MDP is  $20/2 + 3 * (20 * 2) + 2 * 30 + 2 * (30 * 3) + 40/2 + 3 * 40 = 10 + 120 + 60 + 180 + 20 + 120 = 510$  dollars

## 5.5.2 Scheduling Procedure of CTC-MC Algorithm

### Deadline Distribution

According to Equation 1,  $ACT + (n-1) * OT = OD$ , here  $ACT = 2 + 3 + 2 = 7$ ,  $n = 4$  and  $OD = 14$ , we can get  $OT = 2$ .

For the last instance  $I_4$ , we have:

- $sub\_deadline(I_4T_3) = OD = 14$ ,
- $sub\_deadline(I_4T_2) = sub\_deadline(I_4T_3) - estimated\_execution\_time(I_4T_3) = 14 - 2 = 12$ ,
- $sub\_deadline(I_4T_1) = sub\_deadline(I_4T_2) - estimated\_execution\_time(I_4T_2) = 12 - 3 = 9$ .

Knowing the sub-deadlines of the last instance, we now deduce the sub-deadlines of the other instances as described step 1 in Section 5.4.1:

- $sub\_deadline(I_3T_3) = OD = 14$
- $sub\_deadline(I_3T_2) = sub\_deadline(I_4T_2) - (4 - 3) * 2 = 12 - 2 = 10$
- $sub\_deadline(I_3T_1) = sub\_deadline(I_4T_1) - (4 - 3) * 2 = 9 - 2 = 7$
  
- $sub\_deadline(I_2T_3) = OD = 14$
- $sub\_deadline(I_2T_2) = sub\_deadline(I_4T_2) - (4 - 2) * 2 = 12 - 4 = 8$
- $sub\_deadline(I_2T_1) = sub\_deadline(I_4T_1) - (4 - 2) * 2 = 9 - 4 = 5$
  
- $sub\_deadline(I_1T_3) = OD = 14$
- $sub\_deadline(I_1T_2) = sub\_deadline(I_4T_1) - (4 - 1) * 2 = 12 - 6 = 6$
- $sub\_deadline(I_1T_1) = sub\_deadline(I_4T_1) - (4 - 1) * 2 = 9 - 6 = 3$

### Schedule Round 1

In the first round, start tasks such as  $I_1T_1$ ,  $I_2T_1$ ,  $I_3T_1$  and  $I_4T_1$  can be scheduled at first. Because both service 1 (denoted as  $S_1$ ) and service 2 (denoted as  $S_2$ ) are idle,

$service\_available\_time(S_1) = service\_available\_time(S_2) = 0$ , which means they are available from the beginning. The tasks are scheduled order by the incoming sequence:  $I_1T_1, I_2T_1, I_3T_1$  and  $I_4T_1$ .

(1) For task  $I_1T_1$  which has a *sub-deadline* = 3, if it is scheduled on  $S_1$  that has half normal speed and half normal cost, we will get the following outcomes:

- Because  $I_1T_1$  is the start task, so its earliest start time on  $S_1$  (denoted as  $data\_available\_time(I_1T_1|S_1)$ ) is **0 second**.
- $final\_completion\_time(I_1T_1|S_1) = \max(service\_available\_time(S_1), data\_available\_time(I_1T_1|S_1)) + 2 * estimated\_execution\_time(I_1T_1|S_1) = \max(0, 0) + 2 * 2 = 4$  **second**
- $final\_execution\_cost(I_1T_1|S_1) = estimated\_final\_execution\_cost(I_1T_1) / 2 = 20 / 2 = 10$  **dollars**
- The new  $service\_available\_time(S_1) = final\_completion\_time(I_1T_1|S_1) = 4$  **second**

If it is scheduled on  $S_2$  that only has twice normal speed and twice normal cost, we will get the following outcomes:

- $data\_available\_time(I_1T_1|S_2) = 0$  **second**.
- $final\_completion\_time(I_1T_1|S_2) = \max(service\_available\_time(S_2), data\_available\_time(I_1T_1|S_2)) + estimated\_execution\_time(I_1T_1) / 2 = \max(0, 0) + 2 / 2 = 1$  **second**
- $final\_execution\_cost(I_1T_1|S_2) = estimated\_final\_execution\_cost(I_1T_1) * 2 = 20 * 2 = 40$  **dollars**
- The new  $service\_available\_time(S_2) = final\_completion\_time(I_1T_1|S_2) = 1$  **second**

We can see that although the execution cost on  $S_1$  is less, the completion time on  $S_1$  will exceed the sub-deadline. Thus  $I_1T_1$  has to be scheduled to the more expensive  $S_2$ .

(2) For task  $I_2T_1$  which also has a *sub-deadline* = 5, if it is scheduled on  $S_1$  which only has a half normal speed and half normal cost, we will get the following outcomes:

- $data\_available\_time(I_2T_1|S_1) = 0$  **second.**
- $final\_completion\_time(I_2T_1|S_1) = \max(service\_available\_time(S_1), data\_available\_time(I_1T_1|S_1)) + 2 * estimated\_execution\_time(I_1T_1|S_1) = \max(4, 0) + 2 * 2 = 8$  **second**
- $final\_execution\_cost(I_2T_1|S_1) = estimated\_final\_execution\_cost(I_2T_1) / 2 = 20 / 2 = 10$  **dollars**
- The new  $service\_available\_time(S_1) = final\_completion\_time(I_2T_1|S_1) = 8$  **second**

If it is scheduled on  $S_2$  which only has twice normal speed and twice normal cost, we will get the following outcomes:

- $data\_available\_time(I_2T_1|S_2) = 0$  **second.**
- $final\_completion\_time(I_2T_1|S_2) = \max(service\_available\_time(S_2), data\_available\_time(I_1T_1|S_1)) + 2 * estimated\_execution\_time(I_1T_1|S_1) = \max(0, 0) + 2 * 2 = 4$  **second**
- $final\_execution\_cost(I_2T_1|S_2) = estimated\_final\_execution\_cost(I_2T_1) * 2 = 20 * 2 = 40$  **dollars**
- The new  $service\_available\_time(S_2) = final\_completion\_time(I_2T_1|S_2) = 4$  **second**

Because both completion times do not exceed *sub-deadline* ( $I_2T_1$ ) and the execution cost on  $S_1$  is less than that on  $S_2$ ,  $I_2T_1$  is scheduled on  $S_1$  finally.

(3) Following the same routine,  $I_4T_1$  is scheduled on  $S_2$ , while  $I_3T_1$  is scheduled on  $S_2$ . the scheduling result is shown in Figure 5.4.

## Schedule Round 2

In the second round, tasks  $I_1T_2$ ,  $I_2T_2$ ,  $I_3T_2$  and  $I_4T_2$  can be scheduled because their predecessors have already been scheduled. As both service 3 (denoted as  $S_3$ ) and service 4 (denoted as  $S_4$ ) are idle, thus  $service\_available\_time(S_3) = service\_available\_time(S_4) = 0$ , which means they are available from the beginning. We can see from Figure 5.3

that these tasks are available in sequence of  $I_1T_2$ ,  $I_3T_2$ ,  $I_2T_2$  and  $I_4T_2$ , which is determined by the completion time of their respective predecessors. Therefore, the tasks are scheduled in this sequence.

(1) For task  $I_1T_2$  which has a *sub-deadline* = 7, if it is scheduled on  $S_3$  which has exactly the normal speed and exactly the normal cost, we will get the following outcomes:

- Because  $I_1T_2$  has a predecessor  $I_1T_1$ , so  $data\_available\_time(I_1T_2|S_3) = final\_completion\_time(I_1T_1|S_2) + data\_transmission\_time(S_2, S_3) = 1 + 1 = 2$  **second**.
- $final\_completion\_time(I_1T_2|S_3) = \max(service\_available\_time(S_3), data\_available\_time(I_1T_2|S_3)) + estimated\_execution\_time(I_1T_2|S_3) = \max(2, 0) + 3 = 5$  **seconds**
- $final\_execution\_cost(I_1T_2|S_3) = estimated\_final\_execution\_cost(I_1T_2) = 30$  **dollars**
- The new  $service\_available\_time(S_3) = final\_completion\_time(I_1T_2|S_3) = 5$  **second**

If it is scheduled on  $S_4$  which has twice normal speed and twice normal cost, we will get the following outcomes:

- $data\_available\_time(I_1T_2|S_4) = final\_completion\_time(I_1T_1|S_2) + data\_transmission\_time(S_2, S_4) = 1 + 1 = 2$  **second**.
- $final\_completion\_time(I_1T_2|S_4) = \max(service\_available\_time(S_4), data\_available\_time(I_1T_2|S_4)) + estimated\_execution\_time(I_1T_2) / 3 = \max(0, 2) + 3 / 3 = 3$  **second**
- $final\_execution\_cost(I_1T_2|S_4) = estimated\_final\_execution\_cost(I_1T_2) * 3 = 30 * 3 = 90$  **dollars**
- The new  $service\_available\_time(S_4) = final\_completion\_time(I_1T_2|S_4) = 3$  **second**

As both completion times do not exceed *sub-deadline* ( $I_1T_1$ ) and the execution cost on  $S_3$  is less than that on  $S_4$ ,  $I_1T_2$  is scheduled on  $S_3$  finally.

(2) Following the same routine,  $I_2T_2$  is also scheduled on  $S_3$ ,  $I_3T_2$  and  $I_4T_2$  are scheduled on  $S_4$ , the scheduling result is shown in Figure 5.4.

### Schedule Round 3

The tasks are available in sequence of  $I_3T_3$ ,  $I_1T_3$ ,  $I_2T_3$  and  $I_4T_3$ . Finally  $I_3T_3$  and  $I_1T_3$  are scheduled on  $S_5$ ,  $I_2T_3$  and  $I_4T_3$  are scheduled on  $S_6$ , the scheduling result is shown in Figure 5.4.

### Execution Cost

The total cost of Deadline-MDP is  $2 * (20/2) + 2 * (20 * 2) + 2 * 30 + 2 * (30 * 3) + 2 * (40/2) + 2 * 40 = 20 + 80 + 60 + 180 + 40 + 80 = 460$  dollars. Thus the execution cost is less than that of Deadline-MDP algorithm.

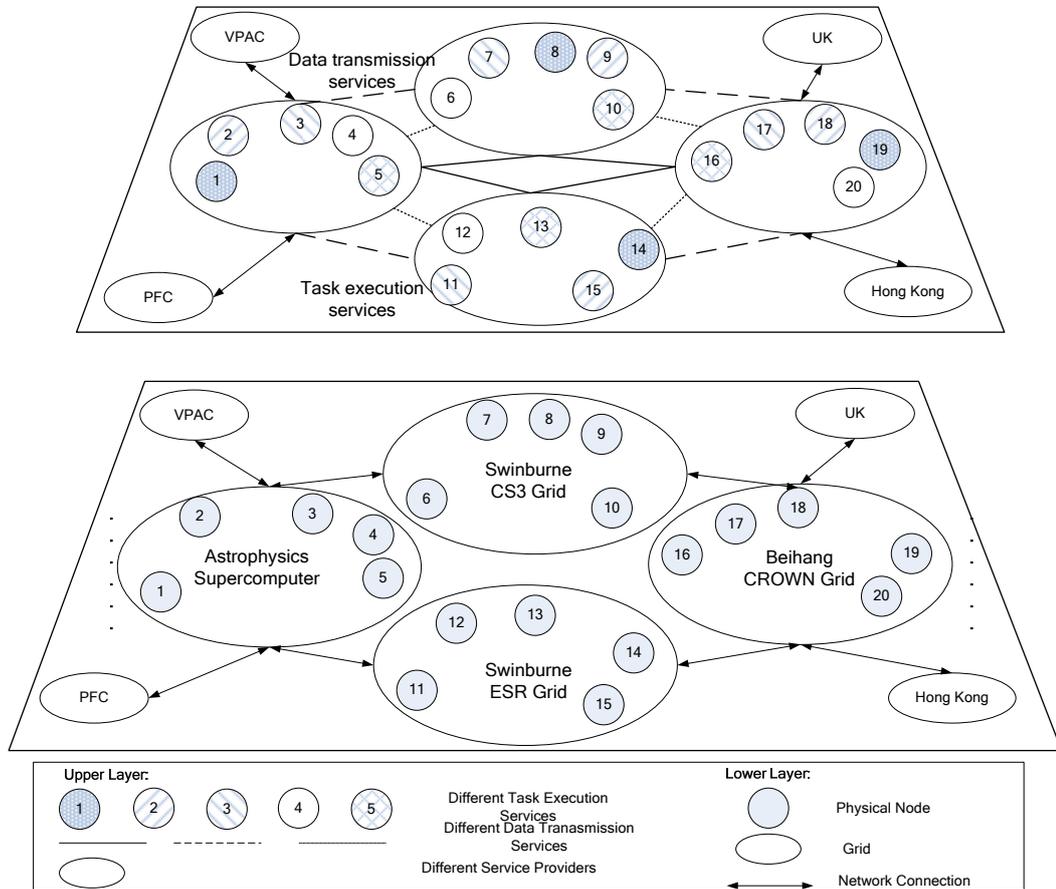
## 5.6 Simulation and Comparison

In this section, we simulate the instance-intensive cost-constrained cloud workflow scenario on our SwinDeW-Cb platform, and present the mean execution cost of concurrent workflow instances using our CTC algorithm in comparison to the other representative scheduling algorithm, namely, the Deadline-MDP algorithm [YBT2005] which is more effective than Deadline-Level and Greedy-Cost algorithms as overviewed in Section 2.

### 5.6.1 Simulation Environment

As shown in Figure 5.5, the SwinDeW-Cb (**Swinburne Decentralised Workflow for Cloud based on Business model**) is also built on top of the Swinburne Workflow Test Environment (SWTE). The upper layer represents the logic infrastructure of scheduling algorithm, while the lower layer represents the SWTE. The numbers in the figure represents the mapping relationship of logic scheduling peers and physical nodes, i.e., if the number on a logic scheduling peer on upper layer is the same with that of a physical

node on lower layer, it implies that this logic scheduling peer is implemented on this physical node.



**Figure 5.5 Swinburne Decentralised Workflow for Cloud based on Business Model (SwinDeW-Cb)**

To simulate the task execution services and data transmission services with different QoS and prices provided by different cloud service providers, it can be seen from Figure 5.5 that each node is modelled as a charged cloud service. Different filling effects represent different types of cloud services which are needed for execution of different tasks, and different line patterns stand for different types of data transmission services which are needed for task data transmission. However, the QoS and prices of these services vary just like that is shown in Table 5.2 and Table 5.3.

The architecture of SwinDeW-Cb is shown in Figure 5.5 with SWTE listed at the bottom for reference.

### 5.6.2 Criteria for Comparison

Makespan (execution time) is the time spent from the beginning of the first task to the end of the last task for a workflow instance. It is normally the most popular criterion for workflow scheduling algorithms. The shorter the mean execution time is, the better the performance is.

As cost is also a very significant factor for cloud workflows, users normally need to pay every time they use cloud services. The workflow systems should be designed to save the cost for users. Usually as long as the completion time does not exceed the deadline, the smaller the cost is, the more satisfied the users are.

In a batch mode scenario using our stream-pipe technique for scheduling instance-intensive cost-constrained cloud workflows, mean execution time and mean execution cost are more suitable for assessing the scheduling outcome. Therefore, we compare our CTC algorithm with Deadline-MDP against the above two criteria.

### 5.6.3 Simulation Process

Once again as mentioned in Section 1.1, instance-intensive cloud workflows are *workflows with a huge number of (hence instance-intensive) concurrent workflow instances, usually much simpler than those complex scientific workflows, enabled on a cloud computing environment (hence cloud workflows)*. To simulate such workflows on business-based cloud computing environment, we construct the simulation environment focusing on the following five aspects. As the first and second aspects are identical to those already illustrated in Section 3.6.3, we list new aspects only as follows:

- In the simulation, there are different types of services for selection. For the purpose of testing, we provide services with different performance based on Table 5.2 and services with different transmission speeds based on Table 5.3 for the algorithms to choose from.
- For fault tolerance testing, we set an execution failure ratio for each task, which means that each task has a minor possibility that it will fail to be completed. If a

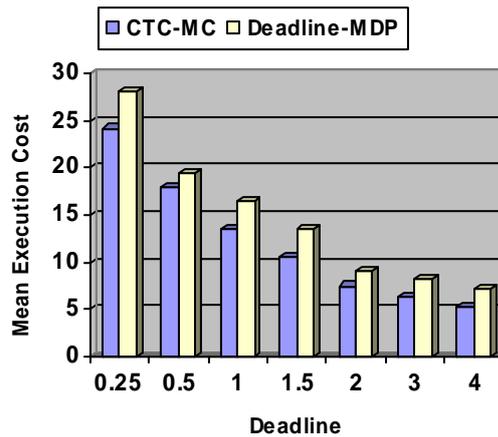
task failed to be completed, it will be processed in high priority in the next scheduling round as described in Section 5.6.4.

- The servers were executing other tasks when we did the simulation, thus they have their dynamic background load accordingly.

The simulation includes two processes. The first calculates the actual mean execution cost within 0.25 to 4 times the input deadline, using CTC-MC and Deadline-MDP algorithms, and the second calculates the actual mean execution time within 0.25 to 4 times the input cost, using CTC-MT and Deadline-MDP algorithms respectively to compare their performance.

#### 5.6.4 Results and Analysis

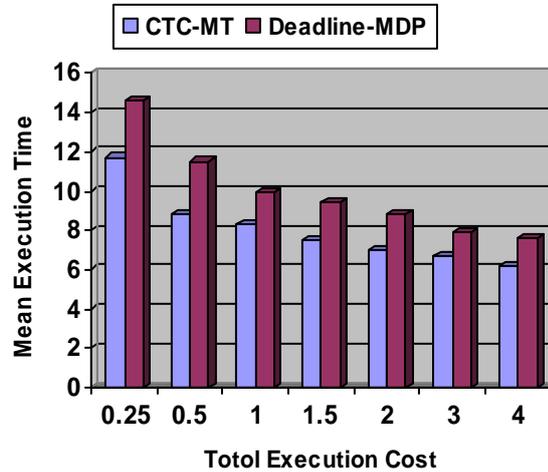
Figure 5.6 demonstrates the comparison results on the mean execution cost within different deadlines. Given both algorithms meet the deadlines, it can be seen that the mean execution cost of our CTC-MC algorithm is always lower than that of the Deadline-MDP algorithm in all circumstances. On average, the saving on the mean execution cost is over 15%.



**Figure 5:6 Comparison on Mean Execution Cost**

Figure 5.7 demonstrates the comparison results of the mean execution time within different execution costs. Given both algorithms do not exceed the execution cost, it can

be seen that the mean execution time of our CTC-MT algorithm is always shorter than that of the Deadline-MDP algorithm in all circumstances. On average, the saving on the mean execution time is over 20%.



**Figure 5.7 Comparison on Mean Execution Time**

## 5.7 Summary

As mentioned in Section 1.2, the most common infrastructures of cloud computing environment consist of physically collocated grids (grid-based), of services scattered on the Internet (service-based) and of charged services (business-based). This chapter mainly focuses on workflow scheduling on a business-based cloud computing environment, while scheduling instance-intensive workflows on the grid-based cloud computing environment and service-based cloud computing environment have already been discussed in Chapter 3 and Chapter 4.

As it is mentioned in Section 1.2.2, it is emergent to design scheduling algorithms for cost-constrained instance-intensive workflows in a commercial cloud computing environment. In a commercial cloud computing environment, customers generally do not own the infrastructure, they merely access or rent, so they can avoid capital expenditure and consume resources as a service. Because users are normally sensitive to execution cost, the cost would be another major concern for cloud workflow applications in addition to execution time.

This chapter has proposed an innovative instance-intensive cost-constraint cloud workflow scheduling algorithm named CTC (compromised-time-cost) which takes cost and time as the main concerns with user intervention enabled on the fly and incorporates the characteristics of cloud computing. The CTC algorithm can either minimise the cost under certain user-designated deadlines (denoted as CTC-MC) or minimise the execution time under certain user-designated cost (denoted as CTC-MT). The simulation has demonstrated that our CTC algorithm can achieve lower cost than others while meeting the user-designated deadline or reduce the mean execution time than others within the user-designated execution cost.

# Chapter 6

## Discussion

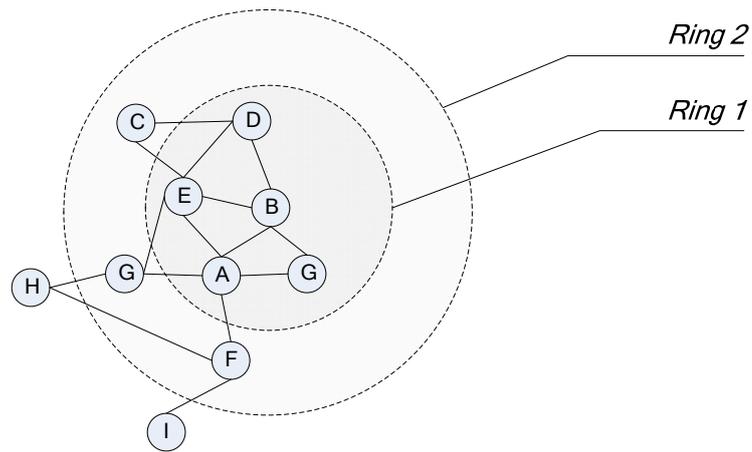
In this chapter, we discuss the possible further improvements on our algorithms. In particular, Section 6.1 proposes an improvement for the OAL algorithm in TMS strategy, Section 6.2 proposes an improvement for the MMA algorithm, and Section 6.3 proposes an improvement for the CTC algorithm. Finally, Section 6.4 summarises the chapter.

### 6.1 Improvements to OAL Algorithm in TMS Strategy

#### 6.1.1 Deficiency of OAL Algorithm in TMS Strategy

As defined in Section 3.4.1, in the OAL algorithm the OAL value is defined as follow: For a specific node  $A$ , the OAL value of its direct neighbour  $B$  is the average load value of  $B$  and all its neighbours except  $A$ . The OAL can present the average load of the opposite direction which can guide the scheduling procedure later.

As shown in Figure 6.1, the OAL value in OAL algorithm is calculated inside *Ring I*, which circles node  $B$  and includes all nodes with a distance 1 from  $B$  if applying Breadth First Search (BFS). The OAL value presents the load information opposite to  $A$  in this area around  $B$ . Although it is more accurate than Gossip algorithm where the area around  $B$  is limited to  $B$  itself, we can count on more accurate load information by expanding the area further



**Figure 6.1 Area for OAL Value Calculation**

### 6.1.2 Suggestion for Improvement

Naturally, we can consider expanding the ring to cover wider area for better guidance. For example, *Ring 2* which includes all nodes with a distance 2 from *B* if applying Breadth First Search (BFS) will provide more accurate load information than *Ring 1*. However, the OAL definition should be modified as follows to reflect the change:

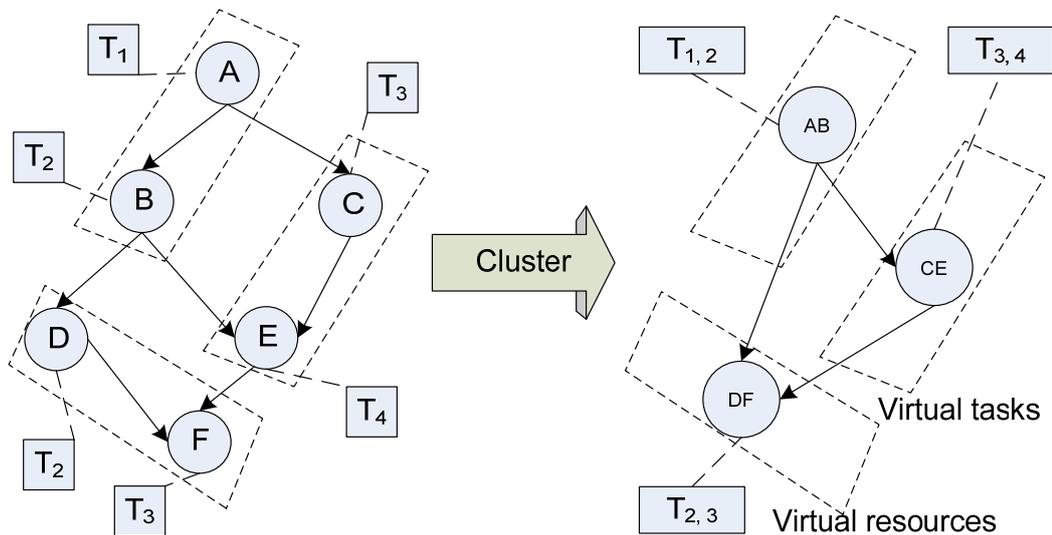
For a specific node *A*, the OAL value of its direct neighbour *B* is the average load value of *B* and all its neighbours, except those neighbours whose distance from *B* is greater than that from *A*. Neighbours that are closer to *A* do not count for the average load in the opposite direction.

However, with the expansion of the ring, the load information needed to calculate OAL value is increasing exponentially, for the calculation now involving those neighbours of neighbours. This will surely increase the scheduling cost. Before apply the change, the compromise should be made between accuracy and scheduling execution cost.

## 6.2 Improvement of MMA Algorithm

### 6.2.1 Deficiency of MMA Algorithm

In workflow scheduling, two tasks located on a directed edge are dependant to each other. In Figure 6.2, a directed edge is denoted as  $(X/T_i, Y/T_j)$ , where  $X$  and  $Y$  are the name of task and  $T_i$  and  $T_j$  are the resource types needed to execute task  $X$  and  $Y$  respectively. Thus we can say resource type  $X$  and resource type  $Y$  have certain affinity. The affinity of two resource types represents how often two successive tasks will use resources of these two resource types, thus the reduction of the communication time between affinitive resources would improve the scheduling performance significantly.



**Figure 6.2 Example of Resource Clustering**

However, this affinity between resource types is not significantly considered in MMA and other similar algorithms. In fact, we can bind some of affinitive resources which are near to each other to virtual resources in advance, thus the minimisation of communication time between successive tasks which use the virtual resources can be ensured.

## 6.2.2 Suggestion for Improvement

### (1) Initialisation of the Affinity Matrix

Virtual tasks are created if the two tasks located on a selected edge. A selected edge is the edge attached to a task node which has most influence of execution time among all edges attached to the node, while each task node can only be attached to one selected edge at one time. For example, the edge located on the critical path or the edge has the largest amount data to transmit. For example, in Figure 6.2, suppose edges  $(A/T_1, B/T_2)$ ,  $(C/T_3, E/T_4)$  and  $(D/T_2, F/T_3)$  are selected, then we can combine task A and task B, task C and task E, task D and task F together as virtual tasks AB, CE and DF respectively.

First, an  $N \times N$  affinity matrix  $M$  is initialised to represent the affinity between different resource types, where  $N$  is the number of resource types. Each row and column of  $M$  represents a resource type and each element  $M_{ij}$  stands for the affinity value of the resource type  $i$  and resource type  $j$ .

At the beginning, the value of the matrix is reset to 0, and for each selected edge of each DAG, suppose that the nodes on the edge need resource type  $i$  and resource type  $j$  to execute respectively,  $M_{ij}$  is increased by  $n$ , where  $n$  is the number of instances generated from the DAG. Finally we will get the affinity matrix with value. Figure 6.3 is a simple example of resource type affinity matrix.

	T1	T2	T3	T4
T1	-	0	1	2
T2	0	80	5	75
T3	45	5	53	3
T4	0	75	3	78

**Figure 6.3 Example of Resource Type Affinity Matrix**

## (2) Clustering of Resource Types

Then we can apply BEA (Bond Energy Algorithm) [OV1991] to the affinity matrix to cluster the related resource types together. The BEA algorithm takes as input the attribute affinity matrix, permutes its rows and columns, and generates a clustered affinity matrix. BEA is performed in an insertion-based manner. First column 1 and 2 are copied to the new matrix. Then for column 3, we calculate the contribution value for each position it may be inserted, namely, before column 1, between column 1 and 2, and after column 2, and choose the position which give the largest contribution as described in BEA. The contribution is calculated as follow:

$$cont(i, k, j) = 2bond(i, k) + 2bond(k, j) - 2bond(i, j) \quad (1)$$

$$bond(x, y) = \sum_{z=1}^n M_{zx}M_{zy} \quad (2)$$

After all columns are inserted to proper position, the rows are organized in the same order as the columns. Figure 6.4 shows an example of BEA procedure. It can be seen that  $T_1$  and  $T_3$ ,  $T_2$  and  $T_4$  are regarded as affinitive resource types because  $M_{1,3}$  and  $M_{2,4}$  have the biggest affinity values in the matrix.

Matrix e:	<table style="border-collapse: collapse; text-align: center;"> <tr><td></td><td>T1</td><td>T2</td></tr> <tr><td>T1</td><td>45</td><td>0</td></tr> <tr><td>T2</td><td>0</td><td>80</td></tr> <tr><td>T3</td><td>45</td><td>5</td></tr> <tr><td>T4</td><td>0</td><td>75</td></tr> </table>		T1	T2	T1	45	0	T2	0	80	T3	45	5	T4	0	75	<table style="border-collapse: collapse; text-align: center;"> <tr><td></td><td>T1</td><td>T3</td><td>T2</td></tr> <tr><td>T1</td><td>45</td><td>45</td><td>0</td></tr> <tr><td>T2</td><td>0</td><td>5</td><td>80</td></tr> <tr><td>T3</td><td>45</td><td>53</td><td>5</td></tr> <tr><td>T4</td><td>0</td><td>3</td><td>75</td></tr> </table>		T1	T3	T2	T1	45	45	0	T2	0	5	80	T3	45	53	5	T4	0	3	75															
	T1	T2																																																		
T1	45	0																																																		
T2	0	80																																																		
T3	45	5																																																		
T4	0	75																																																		
	T1	T3	T2																																																	
T1	45	45	0																																																	
T2	0	5	80																																																	
T3	45	53	5																																																	
T4	0	3	75																																																	
	<table style="border-collapse: collapse; text-align: center;"> <tr><td></td><td>T1</td><td>T3</td><td>T2</td><td>T4</td></tr> <tr><td>T1</td><td>45</td><td>45</td><td>0</td><td>0</td></tr> <tr><td>T2</td><td>0</td><td>5</td><td>80</td><td>75</td></tr> <tr><td>T3</td><td>45</td><td>53</td><td>5</td><td>3</td></tr> <tr><td>T4</td><td>0</td><td>3</td><td>75</td><td>78</td></tr> </table>		T1	T3	T2	T4	T1	45	45	0	0	T2	0	5	80	75	T3	45	53	5	3	T4	0	3	75	78	<table style="border-collapse: collapse; text-align: center;"> <tr><td></td><td>T1</td><td>T3</td><td>T2</td><td>T4</td></tr> <tr><td>T1</td><td>45</td><td>45</td><td>0</td><td>0</td></tr> <tr><td>T3</td><td>45</td><td>53</td><td>5</td><td>3</td></tr> <tr><td>T2</td><td>0</td><td>5</td><td>80</td><td>75</td></tr> <tr><td>T4</td><td>0</td><td>3</td><td>75</td><td>78</td></tr> </table>		T1	T3	T2	T4	T1	45	45	0	0	T3	45	53	5	3	T2	0	5	80	75	T4	0	3	75	78
	T1	T3	T2	T4																																																
T1	45	45	0	0																																																
T2	0	5	80	75																																																
T3	45	53	5	3																																																
T4	0	3	75	78																																																
	T1	T3	T2	T4																																																
T1	45	45	0	0																																																
T3	45	53	5	3																																																
T2	0	5	80	75																																																
T4	0	3	75	78																																																

**Figure 6.4 An Example of BEA Procedure**

### (3) Creation of Virtual Resources

For each pair of resource types that are affinitive, e.g.  $T_1$  and  $T_3$ , we create a number of virtual resources  $T_{1,3}$  which are composed of two resources, one's resource type is  $T_1$  and the other's resource type is  $T_2$ . The number of virtual resources is calculated as follows.

Suppose  $N_i$  is the total number of resources of type  $T_i$ , and  $N_j$  is the total number of resources of type  $T_j$ ,  $P(T_i T_j / T_i)$  be the proportion of edges like  $(X/T_i, Y/T_j)$  or  $(X/T_j, Y/T_i)$  in all edges that include  $T_i$ , and  $P(T_i T_j / T_j)$  be the proportion of edges like  $(X/T_i, Y/T_j)$  or  $(X/T_j, Y/T_i)$  in all edges that include  $T_j$ , the number of virtual resources  $N_{ij}$  is calculated as follows:

$$N_{ij} = \text{Min}(\overline{N_i * P(T_i T_j / T_i)}, \overline{N_j * P(T_i T_j / T_j)}) \quad (3)$$

Then we traverse all resources of type  $T_i$  and  $T_j$ , find the resource  $R_i$  of type  $T_i$  and  $R_j$  of type  $T_j$  that give the minimum communication time and combine them as a virtual resource, then remove them from resource list. The steps repeated until  $N_{ij}$  virtual resources are created. Correspondingly, we combine task  $X, Y$  together and refer to them as a virtual task.

### (4) Using MMA Algorithm to Schedule Tasks on Resources

After virtual resources and virtual tasks are created, we can use any existing workflow scheduling algorithms, for example, the MMA algorithm described in Chapter 4, to schedule tasks (including normal tasks and virtual tasks) on resources (including normal resources and virtual resources).

## 6.3 Improvement of CTC Algorithm

### 6.3.1 Deficiency of CTC Algorithm

The deficiency of the CTC algorithm is that the sub-deadlines are calculated at build time and fixed during run time. However, if some tasks are finished earlier or later than

expected at run time, it is necessary to adjust the deadline of successive tasks dynamically to reflect the change. In detail, when a task is finished earlier, the sub-deadlines of its successive tasks should be adjusted accordingly to take advantage of the time saved, thus cheaper but slower resource can be selected later to decrease the cost further while guarantee the overall deadline. On the contrary, when a task is finished later, the sub-deadlines of its successive tasks should be adjusted accordingly to compensate the extra time, thus more expensive but faster resource will be selected later to guarantee the overall deadline.

### 6.3.2 Suggestion for Improvement

The dynamic adjusting of sub-deadlines during scheduling process can be described as follows.

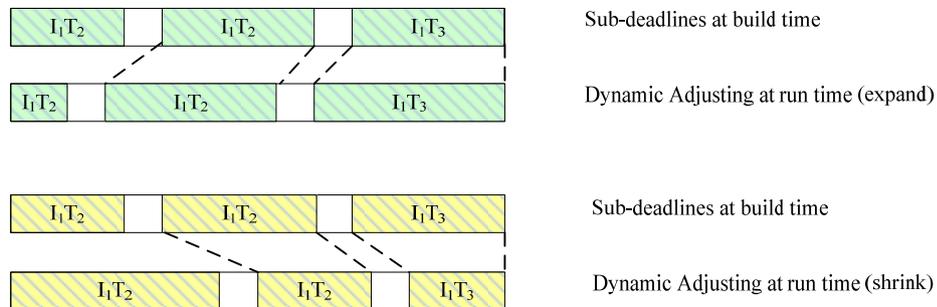
At run time, when a task has been finished, if the difference of its completion time and its sub-deadline is bigger than the threshold, the sub-deadlines of its successive tasks will be adjusted accordingly.

$$TH = OD * 5\% \quad (4)$$

where  $TH$ = threshold,  $OD$ =overall deadline of an instance.5% is the suggested percentage only, other values are also applicable.

$$D = |ACT - SD| \quad (5)$$

where  $D$ = difference,  $ACT$ = actual completion time,  $SD$  = sub-deadline.



**Figure 6.5 Dynamic Adjusting at Run Time**

As Figure 6.5 shows, if task  $I_1T_2$  completed earlier, the distributable execution time of successive tasks  $I_1T_2$  and  $I_1T_3$  can be expanded accordingly to enable the choice of slower but cheaper services, and if task  $I_1T_2$  completed later, the distributable execution time of successive tasks  $I_1T_2$  and  $I_1T_3$  should be shrunk to ensure the overall deadline.

## 6.4 Summary

In this chapter, the possible improvements of the proposed scheduling algorithms are discussed. The first improvement is the expansion of the area where the OAL value is calculated for better guidance for the scheduling algorithms. The second improvement is based on the MMA algorithm. The idea is to combine the nearest affinitive resources to virtual resources in advance to reduce the communication time between the successive tasks of virtual tasks which use the virtual resources. The third improvement is for the CTC algorithm. It dynamically adjusts the sub-deadlines at run time if a certain threshold is exceeded to reduce the execution cost further or ensure the overall deadline.

# Chapter 7

## Conclusions and Future Work

### 7.1 Summary of this Thesis

The objective of this thesis was to design scheduling algorithms for instance-intensive cloud workflows. The thesis was organised as follows.

- Chapter 1 gave a brief introduction to the work of this thesis. First it gave the definition of instance-intensive cloud workflows, the research objective, and then discussed the key issue of designing scheduling algorithms for such workflows in different cloud computing environments.
- Chapter 2 addressed the related work of this thesis. In detail, it first introduced the knowledge of grid computing, cloud computing and their relationship. Then it gave an introduction to workflow and workflow management systems (WMSs). After that, it introduced the workflow scheduling algorithms and classified existing workflow scheduling algorithms into two main types, namely, the best-effort scheduling and QoS-constrained workflow scheduling algorithms, and introduced some typical workflow scheduling algorithms of these two types briefly. This chapter then presented the bank cheque processing scenario and analysed the problems of existing workflow scheduling algorithms in such a scenario, thus new algorithms are needed for scheduling such workflows. Finally it pointed out the requirements of the instance-intensive workflow scheduling algorithms.

- Chapter 3 described the Throughput Maximisation Strategy (TMS) for scheduling instance-intensive cloud workflows in the cloud computing environment based on grid. In detail, this strategy consists of two algorithms: the Opposite Average Load (OAL) algorithm that balances the load at the instance level and the Extended Min-Min (EMM) algorithm that maximises the resource utilization rate at the task level. Also, it introduced the Swinburne Workflow Test Environment (SWTE) and SwinDeW-Cg (**Swinburne Decentralised Workflow for Cloud based on Grid**), the simulation environment used to implement and compare TMS and its reference algorithms. The simulation demonstrated that TMS strategy has shorter mean execution time, higher resource utilisation rate and higher overall throughput than its reference algorithms.
- Chapter 4 described the Min-Min-Average (MMA) algorithm for scheduling instance-intensive cloud workflows in cloud computing environment based on service where considerable communication time involved. It also built SwinDeW-Cs (**Swinburne Decentralised Workflow for Cloud based on Service**), the simulation environment used to implement and compare MMA and its reference algorithms. The simulation demonstrated that MMA algorithm has shorter mean execution time and higher overall throughput than its reference algorithms.
- Chapter 5 described the Compromised-Time-Cost (CTC) algorithm for scheduling instance-intensive cloud workflows in cloud computing environment based on business model. It also built SwinDeW-Cb (**Swinburne Decentralised Workflow for Cloud based on Business Model**), the simulation environment used to implement and compare CTC and its reference algorithms. The simulation demonstrated that CTC can achieve lower execution cost than its reference algorithms within user-designated deadline, or achieve shorter execution time than its reference algorithms within user-designated budget.
- Chapter 6 discussed the possible improvements of our algorithms. The improvements include the expansion of the area where the OAL value of OAL algorithm in the TMS strategy is calculated for better guidance for the

scheduling algorithms, the combination of nearest affinitive resources to virtual resources in advance to reduce the communication time between the successive tasks of virtual tasks which use the virtual resources in the MMA algorithm, and the dynamically adjusting of the sub-deadlines of successive tasks at run time if the difference of actual and estimated completion time of a task exceeds certain threshold to reduce the execution cost further or ensure the overall deadline in the CTC algorithm.

- This chapter gave a summary of the thesis and will conclude the contributions of the thesis, and point out future work.

## 7.2 Contributions of this Thesis

The significance of this research is that it designed a set of scheduling algorithms for instance-intensive cloud workflows. Instance-intensive cloud workflows are workflows with a huge number of workflow instances (hence, instance intensive) enabled on a cloud computing environment (hence cloud workflows). The processing of a huge number of cheques on a daily basis is a typical example of such workflows. However, most existing workflow scheduling algorithms are designed for complicated scientific workflows, and none are dedicated to instance-intensive cloud workflows. In addition, this research considered different requirements for scheduling workflows in different cloud computing environments. In detail, it designed different algorithms for different cloud computing environments of different system infrastructures. Moreover, it proposed three different scheduling infrastructures for each of the proposed scheduling algorithms, which can guide the system design of the SwinDeW-C (**Swinburne Decentralised Workflow for Cloud**) workflow management system. In particular, the major contributions of this thesis are:

- *The proposition of instance-intensive cloud workflows.*

This thesis has proposed the concept of instance-intensive workflows which can be used to model some e-business processes such as the bank cheque processing. Moreover, it has introduced cloud computing to facilitate the execution of such workflows.

- *An innovative strategy for scheduling instance-intensive workflows in a grid-based cloud computing environment.*

This thesis has proposed an innovative strategy called Throughput Maximisation Strategy (TMS) for scheduling instance-intensive workflows in a grid-based cloud computing environment. This strategy divides the scheduling into two stages according to the system infrastructure of the grid-based cloud computing environment. At the first stage, workflow instances are balanced among grids by the Opposite Average Load (OAL) algorithm, and at the second stage, the utilisation rate of resources within each autonomous grid is maximised by the Extended Min-Min (EMM) algorithm. With the strategy, the overall throughput is significantly increased.

- *An innovative algorithm for scheduling instance-intensive workflows in a service-based cloud computing environment.*

This thesis has proposed an innovative algorithm called Min-Min-Average (MMA) for scheduling instance-intensive workflows in a service-based cloud computing environment. Unlike the grid-based cloud computing environment where the communication time within each grid is usually negligible, the communication time between services scattered on the Internet can no longer be neglected. Thus how to reduce the communication time becomes an important issue. The MMA algorithm first establishes a fundamental infrastructure that can provide nearest neighbours to decrease the communication time significantly, and then uses a scheduling strategy which can increase the utilisation rate of both the execution unit and communication unit of the workflow execution engine to achieve the maximum overall throughput.

- *An innovative algorithm for scheduling instance-intensive workflows in a business-based cloud computing environment.*

This thesis has proposed an innovative algorithm called Compromised-Time-Cost (CTC) for scheduling instance-intensive workflows in a business-based cloud computing environment. In a commercial cloud computing environment, users need not to possess any services but instead pay every time they use the required services (pay per use). However, the execution time and execution cost are always a conflicting. CTC actually consists of two algorithms, namely, the

CTC-MC (Compromised-Time-Cost algorithm for Minimising execution Cost) algorithm, which minimises the execution cost within user designated deadline, and the CTC-MT (Compromised-Time-Cost algorithm for Minimising execution Time) algorithm, which minimises the execution time within user designated budget. Each algorithm can enable users to input a compromise of time and cost on the fly to achieve maximum user satisfaction.

- *Some intuitive suggestions for improvement of instance-intensive cloud workflow scheduling algorithms.*

Besides proposing three innovative algorithms for scheduling instance-intensive workflows in different cloud computing environments, this thesis also presented some the suggestions for improvement. The first suggestion is the extension of the OAL algorithm with the TMS strategy. It tries to increase the breadth of neighbourhood to increase the accuracy of load estimation. The second suggestion is the proposition of virtual resources and virtual tasks in the MMA algorithm. This suggestion combines those resources of affinitive resource types together to virtual resources. The successive tasks that use the resources that compose the virtual resources are referred to as virtual tasks and can be scheduled directly on the created virtual resources. Because the virtual resources are created among those nearest resources in advance, the communication time can surely be decreased among those successive tasks that use these affinitive resources. The third suggestion is the dynamically adjusting of sub-deadlines in the CTC algorithm. If the difference of actual and estimated completion time of a task exceeds certain threshold, the sub-deadlines of its successive tasks will be adjusted to reduce the execution cost further or ensure the overall deadline in the CTC algorithm.

- *The innovative design of cloud workflow simulation environment (SwinDeW-C family).*

The other contribution of the thesis is the design of cloud workflow simulation environments. In particular, the cloud workflow simulation environment includes SwinDeW-Cg (Swinburne Decentralised Workflow for Cloud based on Grid), SwinDeW-Cs (Swinburne Decentralised Workflow for Cloud based on

Service) and SwinDeW-Cb (Swinburne Decentralised Workflow for Cloud based on Business Model).

### **7.3 Future work**

In the future, further investigation into scheduling algorithms for instance-intensive cloud workflows will be carried out. Future research includes further investigation into cloud computing and the design of new algorithms for scheduling instance-intensive workflows in new cloud environments. Further improvements to the existing cloud workflow scheduling algorithms, for example, the suggestions proposed in Chapter 6, integration and coordination of the scheduling algorithms with other workflow modules, and etc.

Because of the promotion of cloud computing by world's leading companies, more and more cloud computing environments will surely emerge. These cloud computing may have different system infrastructures other than grid-based, service-based or business-based which have already been described in this thesis before, and their requirements for scheduling algorithms will also be different. Therefore, it is necessary to design new scheduling algorithms dedicated to cloud workflows executed on these new cloud computing environments.

Further improvements of the existing cloud workflow scheduling algorithms include more suitable system infrastructure for the scheduling algorithm, more accurate estimation of service load, more accurate prediction of estimated execution time, more efficient management of service usage, more advanced scheduling strategies to increase the overall throughput, and etc.

Moreover, more theoretical analyses of our existing and successive scheduling algorithms for instance-intensive cloud workflows will be performed. This issue will include such as mathematical model of cloud computing environment, instance-intensive workflow tasks and instances, and systematic definition of scheduling process.

Finally, integration and coordination of the scheduling algorithms with other workflow modules include the realisation of the proposed scheduling infrastructures in

workflow management system, the implementation of the scheduling algorithms on top of these scheduling infrastructures, the definitions of interfaces between scheduling algorithms and other workflow modules, especially the coordination between scheduling algorithms and data management in cloud computing.

# Bibliography

- [ABBC+2002] B. Allcock, J. Bester, J. Bresnahan, A. L. Chervenak, I. Foster, C. Kesselman, S. Meder, V. Nefedova, D. Quesnal, S. Tuecke. Data Management and Transfer in High Performance Computational Grid Environments, *Parallel Computing Journal*, 28 (5), 749-771, May 2002.
- [ABJJ+2004] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher and S. Mock, Kepler: An Extensible System for Design and Execution of Scientific Workflows, *Proc. of 16th International Conference on Scientific and Statistical Database Management (SSDBM 2004)*, 423-424, Santorini Island, Greece, June 2004.
- [AF2008] A. Avanes and J. Freytag, Adaptive Workflow Scheduling under Resource Allocation Constraints and Network Dynamics. *Proc. VLDB Endow*, 1(2), 1631-1637, August 2008.
- [AFGJ+2009] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin and Ion Stoï, Above the Clouds: A Berkeley View of Cloud Computing, Technical Report, UCB/EECS-2009-28, UC Berkeley Reliable Adaptive Distributed Systems Laboratory, EECS Department, University of California, Berkeley, February, 2009.
- [All2001] R. Allen, Workflow: an Introduction, *Workflow Handbook*, 2001, Workflow Management Coalition.

- [AK2005] A. K. Aggarwal and R. D. Kent, An Adaptive Generalized Scheduler for Grid Applications, Proc. of the 19th Annual International Symposium on High Performance Computing Systems and Applications (HPCS'05), 15-18, Guelph, Ontario Canada, May 2005.
- [AMGA+95] G. Alonso, C. Mohan, R. Guenthoer, D. Agrawal, A. El Abbadi and M. Kamath, Exotica/FMQM: A Persistent Message-Based Architecture for Distributed Workflow Management, Proc. of IFIP Working Conference on Information Systems for Decentralized Organizations, 1-18, Trondheim, Norway, August 1995.
- [AS1999] J. Almond and D. Snelling, UNICORE: Uniform Access to Supercomputing as an Element of Electronic Commerce, Future Generation Computing Systems 15(5-6), 539-548, October, 1999.
- [BA2004] R. Bajaj and D. P. Agrawal, Improving Scheduling of Tasks in A Heterogeneous Environment, IEEE Transactions on Parallel and Distributed Systems, 15(2), 107-118, February 2004.
- [BAG2000] R. Buyya, D. Abramson, J. Giddy, Nimrod/G: An Architecture for a Resource Management and Scheduling System in a Global Computational Grid, Proc. of The Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region, 2000, 283-289, Beijing, China, May 2000.
- [BCCD+2001] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon and R. Wolski, The GrADS Project: Software Support for High-Level Grid Application Development, International Journal of High Performance Computing Applications (JHPCA), 15(4), 327-344, Winter 2001.

- [BCCM+2006] R. Bindal, P. Cao, W. Chan, J. Medved, J. Suwala, T. Bates and A. Zhang, Improving Traffic Locality in BitTorrent via Biased Neighbour Selection. Proc. of 26th IEEE International Conference on Distributed Computing Systems, 66-80, Lisbon, Portugal, July, 2006.
- [BDGJ+2004] G. B. Berriman, E. Deelman, J. C. Good, J. C. Jacob, D. S. Katz, C. Kesselman, A. C. Laity, T. A. Prince, G. Singh, and M. Su, Montage: a Grid-Enabled Engine for Delivering Custom Science-Grade Mosaics on Demand, Proc. of the International Society for Optical Engineering, 5493(1), 221-232, Kiev, Ukraine, May 2004.
- [BHKL+2000] B. Bode, D. M. Halstead, R. Kendall, Z. Lei, W. Hall, D. Jackson. The Portable Batch Scheduler and the Maui Scheduler on Linux Clusters, Proc. of 4th Annual Linux Showcase & Conference, 27-34, Atlanta, Georgia, USA. October, 2000.
- [BJDG+2005] J. Blythe; S. Jain; E. Deelman; Y. Gil; K. Vahi; A. Mandal and K. Kennedy, Task Scheduling Strategies for Workflow-Based Applications in Grids. Proc. of 2005 IEEE International Symposium on Cluster Computing and the Grid, 2, 759-767, Cardiff, Wales, UK, May 2005.
- [BMQL2007] G. Boss, P. Malladi, D. Quan, L. Legregni and H. Hall, Cloud Computing (White Paper), IBM, October 2007, [http://download.boulder.ibm.com/ibmdl/pub/software/dw/wes/hipods/Cloud\\_computing\\_wp\\_final\\_8Oct.pdf](http://download.boulder.ibm.com/ibmdl/pub/software/dw/wes/hipods/Cloud_computing_wp_final_8Oct.pdf), accessed on May. 19, 2009.
- [Bra2008] R. Bragg, Cloud Computing: When Computers Really Rule, Tech News World, July 2008. Electronic Magazine, available at <http://www.technewsworld.com/story/63954.html>, accessed on May. 19, 2009.

- [BSB2001] T. D. Braun, H. J. Siegel and N. Beck, A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *Journal of Parallel and Distributed Computing*, 61, 810-837, 2001.
- [BV2004] R. Buyya and S. Venugopal, The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report, Proc. of 1st IEEE International Workshop on Grid Economics and Business Models (GECON 2004), 19-36, Seoul, Korea, April 2004.
- [BYV2008] R. Buyya, C. S. Yeo, and S. Venugopal, Market-Oriented Cloud and Atmospheric Computing: Hype, Reality, and Vision, Proc. of 10th IEEE International Conference on High Performance Computing and Communications (HPCC-08), 5-13, Dalian, China, September, 2008.
- [CGHM+2005] D. Churches, G. Gombas, A. Harrison, J. Maassen, C. Robinson, M. Shields, I. Taylor and I. Wang, Programming Scientific and Distributed Workflow with Triana Services, *Concurrency and Computation: Practice and Experience*, 18(10), 1021–1037, December 2005.
- [CGN96] T. Cai, P. A. Gloor and S. Nog, DartFlow: A Workflow Management System on the Web using Transportable Agents, Technical Report PCS-TR96-283, Dartmouth College, 1996.
- [CJSN2003] J. Coa, S. Jarvis, S. Saini, and G. Nudd, GridFlow: Workflow Management for Grid Computing, Proc. of 3rd International Symposium on Cluster Computing and the Grid, 198-205, Tokyo, Japan, May 2003.

- [CZ2009] W. Chen, J. Zhang, An Ant Colony Optimization Approach to a Grid Workflow Scheduling Problem With Various QoS Requirements, IEEE Transactions on Systems, Man, and Cybernetics - Part C: Applications and Reviews, Vol. 39, No. 1, January 2009.
- [DA2007] F. Dong and S. G. Akl, An Adaptive Double-layer Workflow Scheduling Approach for Grid Computing. Proc. of the 21st International Symposium on High Performance Computing Systems and Applications, 7-13, Saskatoon, Canada, May 2007.
- [DBGK+2003] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta and K. Vahi, Mapping Abstract Complex Workflows onto Grid Environments, Journal of Grid Computing, 1, 25-39, 2003.
- [DL2008] Y. Du, X. Li, Application of Workflow Technology to Current Dispatching Order System, International Journal of Computer Science and Network Security, 8(3), 59-61. March 2008.
- [DSSB+2005] E. Deelman, G. Singh, M. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob and D. S. Katz, Pegasus: A Framework for Mapping Complex Scientific Workflows onto Distributed Systems. Scientific Programming, 13(3), 219-237, July 2005.
- [FBAK+2003] L. Ferreira, V. Berstis, J. Armstrong, M. Kendzierski, A. Neukoetter, M. Takagi, R. Bing-Wo, A. Amir, R. Murakawa, O. Hernandez, J. Magowan and N. Bieberstein, Introduction to Grid Computing with Globus, IBM Redbook, available at <http://www.redbooks.ibm.com/redbooks/pdfs/sg246895.pdf>, accessed on May. 19, 2009.

- [FG2006] G. C. Fox and D. Gannon, Workflow in Grid Systems, Concurrency and Computation: Practice and Experience, 18(10), 1009-1019, 2006.
- [FJPP+2005] T. Fahringer, A. Jugravu, S. Pillana, R. Prodan, C. Seragiotto Jr. and H. L. Truong, ASKALON: A Tool Set for Cluster and Grid Computing, Concurrency and Computation: Practice and Experience, 17(2-4), 143-169, 2005.
- [FK1997] I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, Intl Journal of Supercomputer Applications, 11(2), 115-128, 1997.
- [FKLL+1999] I. Foster, C. Kesselman, C. Lee, R. Lindell, K. Nahrstedt and A. Roy, A Distributed Resource Management Architecture that Supports Advance Reservations and Co-Allocation, Proc. of 7th International Workshop on Quality of Service, 27-36, London, UK, May 1999.
- [FKPT+2003] I. Foster, C. Kesselman, L. Pearlman, S. Tuecke, and V. Welch. The Community Authorization Service: Status and Future, Proc. of Computing in High Energy and Nuclear Physics, 24-28, La Jolla, California, USA, March 2003.
- [Fos2002] I. Foster, What is the Grid? A Three Point Checklist, GRIDtoday, Vol. 1, No. 6, July 22, 2002, available at <http://www.mcs.anl.gov/~itf/Articles/WhatIsTheGrid.pdf>, accessed on May. 19, 2009.
- [FTFL+2002] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A Computation Management Agent for Multi-Institutional Grids, Cluster Computing, 5(3), 237-246. 2002.

- [FZRL2008] I. Foster, Y. Zhao, I. Raicu and S. Lu, Cloud Computing and Grid Computing 360-Degree Compared, Grid Computing Environments Workshop, 2008. (GCE '08), 1-10, Austin, Texas, USA, November 2008.
- [GACT+97] E. Gokkoca, M. Altinel, R. Cingil, E. N. Tatbul, P. Koksals and A. Dogac, Design and Implementation of a Distributed Workflow Enactment Service, Proc. of the 2nd IFCIS Conference on Cooperative Information Systems (CoopIS'97), 89-98, South Carolina, USA, June 1997.
- [GAHM98] J. Grundy, M. Apperley, J. Hosking, and W. Mugridge, A Decentralised Architecture for Software Process Modelling and Enactment. IEEE Internet Computing, 2(5), 53-62, September/October 1998.
- [Gee2008] J. Geelan, Twenty One Experts Define Cloud Computing. Virtualization, August 2008, Electronic Magazine, available at <http://virtualization.sys-con.com/node/612375>, accessed on May. 19, 2009.
- [GK2008] G. Gruman and E. Knorr. What Cloud Computing Really Means, InfoWorld, April 2008, Electronic Magazine, available at [http://www.infoworld.com/article/08/04/07/15FE-cloud-computing-reality\\_1.html](http://www.infoworld.com/article/08/04/07/15FE-cloud-computing-reality_1.html), accessed on May. 19, 2009.
- [Glo2005] The Globus Security Team, Grid Security Infrastructure: A Standards Perspective, Technical Report, Argonne National Laboratory, MCS, 2005.

- [HMFD+2008] C. Hoffa, G. Mehta, T. Freeman, E. Deelman, K. Keahey, B. Berriman and J. Good, On the Use of Cloud Computing for Scientific Workflows, Proc. of Fourth IEEE International Conference on e-Science, 640-645, Indianapolis, Indiana, USA, December 2008.
- [IBM1997] IBM International Technical Support Organization, Image and Workflow Library: ImagePlus MVS/ESA Performance and Capacity Planning Benchmark, IBM REDBOOK, available at <http://www.redbooks.ibm.com/redbooks/pdfs/sg244975.pdf>, accessed on May. 19, 2009.
- [KTF2003] N. Karonis, B. Toonen, and I. Foster. MPICH-G2: A Grid-Enabled Implementation of the Message Passing Interface. Journal of Parallel and Distributed Computing, 2003, 63(5):551-563. Academic Press, Inc. Orlando, FL, USA.
- [KTML+2008] D. Kyriazis, K. Tserpes, A. Menychtas, A. Litke and T. Varvarigou, An Innovative Workflow Mapping Mechanism for Grids in the Frame of Quality of Service, Future Generation Computation Systems 24(6), 498-511, June 2008.
- [LABH+2006] B. Ludcher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao and Y. Zhao, Scientific Workflow Management and the Kepler System, Concurrency and Computation: Practice and Experience, 18(10), 1039-1065, 2006.
- [LAHZ+2004] G. von Laszewski, K. Amin, M. Hategan, N. J. Zaluzec, S. Hampton and A. Rossi, GridAnt: A Client-Controllable Grid Workflow System, Proc. of 37th Hawaii International Conference on System Sciences (HICSS-37), 210-219, Hawaii, USA, January 2004.

- [LBRV2005] A. Luther, R. Buyya, R. Ranjan, and S. Venugopal, Peer-to-Peer Grid Computing and a .NET-based Alchemi Framework, High Performance Computing: Paradigm and Infrastructure, Laurence Yang and Minyi Guo (editors), ISBN: 0-471-65471-X, Wiley Press, New Jersey, USA, June 2005.
- [LCJY2009] K. Liu, J. Chen, H. Jin and Y. Yang, A Min-Min Average Algorithm for Scheduling Transaction-Intensive Grid Workflows, Proc. of 7th Australasian Symposium on Grid Computing and e-Research, 41-48, Wellington, New Zealand, January 2009.
- [LCYJ2008] K. Liu., J. Chen, Y. Yang and H. Jin, A Throughput Maximization Strategy for Scheduling Transaction Intensive Workflows on SwinDeW-G, Concurrency and Computation: Practice and Experience, 20(15), 1807-1820, October 2008.
- [LH2005] G. von Laszewski and M. Hategan, Java CoG Kit Karajan/GridAnt Workflow Guide, Technical Report, Argonne National Laboratory, Argonne, Illinois, USA, 2005.
- [MASH+1999] M. Maheswaran, S. Ali, H. J Siegel, D. Hensgen, and R. F. Freund, Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems, Journal of Parallel and Distributed Computing, 59( 2), 107-131, 1999.
- [MC2004] D. A. Menasc and E.Casalicchio, A Framework for Resource Allocation in Grid Computing, Proc. of the 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, 259-267, Volendam, The Netherlands, October, 2004.

- [OAFM+2004] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver and K. Glover, M. R. Pocock, A. Wipat, and P. Li, Taverna: A Tool for the Composition and Enactment of Bioinformatics Workflows, *Bioinformatics*, 20(17), 3045-3054, 2004.
- [Ore2007] T. O'Reilly, What is Web 2.0: Design Patterns and Business Models for the Next Generation of Software, *Communications & Strategies*, 65(1), 17-37, 2007.
- [OV1991] M. T. Ozsu and P. Valduriez, *Principles of Distributed Database Systems*, Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1991.
- [PCVR+2008] S. Pandey, J. Chao, W. Voorsluys, M. Rahman, R. Buyya, Gridbus Workflow Management System on Clouds and Global Grids, *Proc. of IEEE 4th International Conference on e-Science*, 1(1), 323-324, December 2008.
- [PPC97] S. Paul, E. Park and J. Chaar, RainMan: A Workflow System for the Internet, *Proc. of USENIX Symposium on Internet Technologies and Systems*, 159–170, Monterey, California, USA, December 1997.
- [PR2002] L. S. Pitsoulis and M. G. C. Resende, Greedy Randomized Adaptive Search Procedures, P. M. Pardalos and M. G. C. Resende editors, *Handbook of Applied Optimization*, Oxford University Press, 168–182, 2002.
- [RG1999] A. Radulescu and A. J. C. van Gemund, On the Complexity of List Scheduling Algorithms for Distributed Memory Systems, *Proc. of 13th International Conference on Supercomputing*, 68-75, Portland, Oregon, USA, November 1999.

- [RRD03] M. Reichert, S. Rinderle and P. Dadam, ADEPT Workflow Management System: Flexible Support For Enterprise-wide Business Processes (Tool Presentation, Lecturer Notes in Computer Science, 2678, 371-379, Eindhoven, Netherlands, June 2003.
- [RVB2007] M. Rahman, S.Venugopal, R. Buyya, A Dynamic Critical Path Algorithm for Scheduling Scientific Workflow Applications on Global Grids, Proc. of the 3rd IEEE International Conference on e-Science and Grid Computing, 35-42, Bangalore, India, December 2007.
- [RZDF+2007] I. Raicu, Y. Zhao, C. Dumitrescu, I. Foster and M. Wilde, Falcon: a Fast and Light-weight Task Execution Framework, Proc. of the 2007 ACM/IEEE Conference on Supercomputing, 1-12, Seattle, Washington, USA, June, 2007.
- [SCJH+2004] D. P. Spooner, J. Cao, S. A. Jarvis, L. He, and G. R. Nudd. Performance-Aware Workflow Management for Grid Computing, The Computer Journal, 48(3), 347-357, 2005.
- [SRPM+2006] J. M. Schopf, I. Raicu, L. Pearlman, N. Miller, C. Kesselman, I. Foster, M. D'Arcy, Monitoring and Discovery in a Web Services Framework: Functionality and Performance of Globus Toolkit MDS4, Technical Report, Argonne National Laboratory, MCS Preprint #ANL/MCS-P1315-0106, 2006.
- [SZTD2005] R. Sakellariou, H. Zhao, E. Tsiakkouri, and M. D. Dikaiakos., Scheduling Workflows with Budget Constraints, CoreGRID Workshop on Integrated research in Grid Computing, Technical Report TR-05-22, University of Pisa, Dipartimento Di Informatica, Pisa, Italy, November 2005.

- [TTL2005] D. Thain, T. Tannenbaum and M. Livny, Distributed Computing in Practice: The Condor Experience, Concurrency and Computation: Practice and Experience, 17(2-4), 323-356, February-April, 2005.
- [Ull1975] J. D. Ullman, NP-Complete Scheduling Problems, Journal of Computer and System Sciences, 10, 384-393, 1975.
- [VRCL2008] L. M. Vaquero, L. Rodero-Merino, J. Caceres and M. Lindner, A Break in the Clouds: Towards a Cloud Definition, ACM SIGCOMM Computer Communication Review, 39(1), 50-55, December 2008.
- [WfMC99] Workflow Management Coalition, Workflow Management Coalition Terminology & Glossary, February 1999.
- [WPF2005] M. Wiczorek, R. Prodan and T. Fahringer, Scheduling of Scientific Workflows in the ASKALON Grid Environment, Special Issues on Scientific Workflows, ACM SIGMOD Record, 34(3), 56-62, 2005.
- [WSRM1997] L. Wang, H. J. Siegel, V. P. Roychowdhury, and A. A. Maciejewski, Task Matching and Scheduling in Heterogeneous Computing Environments Using a Genetic-Algorithm-Based Approach, Journal of Parallel and Distributed Computing, 47, 8-22, 1997.
- [WTKC+2008] L. Wang, J. Tao, M. Kunze, A. C. Castellanos, D. Kramer and W. Karl, Scientific Cloud Computing: Early Definition and Experience, Proc. of the 10th IEEE International Conference on High Performance Computing and Communications, 825-830, Washington, DC, USA, September, 2008.

- [XCY2007] Z. Xiao, H. Chang and Y. Yi, Optimization of Workflow Resources Allocation with Cost Constraint, Lecture Notes in Computer Science, 4402, 647-656, 2007.
- [YB2004] J. Yu, and R. Buyya, A Novel Architecture for Realizing Grid Workflow Using Tuple Spaces, Proc. of the 5th IEEE/ACM International Workshop on Grid Computing, 119-128, Pittsburgh, USA, November, 2004.
- [YB2005] J. Yu and R. Buyya, A Taxonomy of Workflow Management Systems for Grid Computing, Journal of Grid Computing, 3, 171-200, September 2005.
- [YB2006] J. Yu and R. Buyya, Scheduling Scientific Workflow Applications with Deadline and Budget Constraints using Genetic Algorithms, Scientific Programming Journal, 14(3-4), 217-230, IOS Press, 2006.
- [YB2007] J. Yu and R. Buyya, Workflow Scheduling Algorithms for Grid Computing, Technical Report, GRIDS-TR-2007-10, Grid Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, May 2007.
- [YBT2005] J. Yu, R. Buyya and C. K. Tham, A Cost-based Scheduling of Scientific Workflow Applications on Utility Grids, Proc. of the 1st IEEE International Conference on e-Science and Grid Computing, 140-147, Melbourne, Australia, December 2005.
- [YLCL+2007] Y. Yang, K. Liu, J. Chen, J. Lignier and H. Jin, Peer-to-Peer Based Grid Workflow Runtime Environment of SwinDeW-G, Proc. of the 3rd IEEE International Conference on e-Science and Grid Computing, 51-58, Bangalore, India, December 2007.

- [YLCL+2008] Y. Yang, K. Liu, J. Chen, X. Liu, D. Yuan and H. Jin, An Algorithm in SwinDeW-C for Scheduling Transaction-Intensive Cost-Constrained Cloud Workflows, Proc. of 4th IEEE International Conference on e-Science, 374-375, Indianapolis, USA, December 2008.
- [YMND2003] L. Young, S. McGough, S. Newhouse, and J. Darlington, Scheduling Architecture and Algorithms within the ICENI Grid Middleware, UK e-Science All Hands Meeting, 5-12, IOP Publishing Ltd, Nottingham, UK, September 2003.
- [YYR2006] J. Yan, Y. Yang and G. K. Raikundalia, SwinDeW-a P2P-based Decentralized Workflow Management System, IEEE Transactions on Systems, Man and Cybernetics, Part A, 36(5), 922-935, 2006.
- [ZWFV+2005] Y. Zhao, M. Wilde, I. Foster, J. Voeckler, J. Dobson, E. Gilbert, T. Jordan and E. Quigg, Virtual Data Grid Middleware Services for Data-Intensive Science, Concurrency and Computation: Practice and Experience, 18(6), 595-608, 2005.