

# An Effective Framework of Light-Weight Handling for Three-Level Fine-Grained Recoverable Temporal Violations in Scientific Workflows

Xiao Liu<sup>1</sup>, Zhiwei Ni<sup>2</sup>, Zhangjun Wu<sup>2,1</sup>, Dong Yuan<sup>1</sup>, Jinjun Chen<sup>1</sup>, Yun Yang<sup>1</sup>

<sup>1</sup>*Faculty of Information and Communication Technologies, Swinburne University of Technology  
Hawthorn, Melbourne, Australia 3122  
{xliu, dyuan, jchen, yyang}@swin.edu.au*

<sup>2</sup>*School of Management, Institute of Intelligent Management, Hefei University of Technology  
Hefei, Anhui, China 230009  
{nzwgd, wuzhangjun}@hfut.edu.cn*

**Abstract**—Temporal violations may often take place and deteriorate the overall QoS of scientific workflows. To handle temporal violations in an automatic and cost-effective fashion, we need to resolve the following issues: 1) how to define fine-grained recoverable temporal violations; 2) which light-weight effective exception handling strategies to be facilitated. This paper proposes an effective exception handling framework. Based on a probability based temporal consistency model, the probability range for recoverable temporal violations is divided into three levels of fine-grained temporal violations. Afterwards, three corresponding light-weight exception handling strategies including TDA (Time Deficit Allocation), ACOWR (Ant Colony Optimisation based two-stage Workflow local Rescheduling) and TDA+ACOWR (the combined strategy of TDA and ACOWR) are presented. The experimental results demonstrate the excellent performance of our framework in reducing both local and global temporal violations.

**Keywords**—Temporal Violations; Exception Handling; Workflow Scheduling; Workflow QoS; Scientific Workflows

## 1. Introduction

Scientific workflow systems can be seen as a type of high-level middleware services for high performance computing infrastructures such as grid and cloud computing [1, 8]. Scientific workflows usually underlie many complex e-science applications such as climate modelling, disaster recovery simulation, astrophysics and high energy physics [5]. In recent years, due to the growing demand for high performance computing infrastructures and large scale distributed and collaborative e-science applications, scientific workflows have been attracting increasing interests from distributed and parallel system researchers in the area of High Performance Computing. One of the research issues is how to deliver satisfactory workflow system QoS (quality of service), i.e. how to satisfy workflow QoS constraints such as the constraints on time, cost, fidelity, reliability and security. Among them, time is the basic measurement for system performance and hence attracts many researchers in the workflow area [3, 8, 9, 11].

In reality, a scientific workflow and its workflow segments are normally subject to specific temporal constraints (such as global temporal constraints (deadlines) for workflow instances, and local temporal constraints (milestones) for workflow segments) in order to achieve predefined scientific goals on schedule. Otherwise, the timeliness of its execution results will be significantly deteriorated. For example, a daily weather forecast scientific workflow has to be finished before the broadcasting of the weather forecast program everyday at, for instance, 6:00pm. To deliver satisfactory temporal QoS, the violations of both local and global temporal constraints (or local and global violations for short) need to be proactively detected and handled along scientific workflow execution. Current work on temporal verification in scientific workflows mainly focuses on run-time checkpoint selection [3] and temporal verification [2] which deal with the detection of temporal violations. However, a significant follow-up issue is on how to handle those temporal violations. Specifically, the two fundamental requirements for handling temporal violations are *Automation* and *Cost-effectiveness*.

1) *Automation*. Due to the complex nature of e-science applications and their distributed running environments such as grid and cloud computing, a large number of temporal violations may often be expected in scientific workflows. Besides, since scientific workflows are designed to be highly automatic to conduct large scale distributed scientific processes, human interventions should be avoided as much as possible, especially during workflow run time. Therefore, exception handling strategies are required to be fully automatic so as to reduce the effect of temporal violations on workflow execution and relieve system users from the heavy workload of handling those exceptions.

2) *Cost-effectiveness*. Handling temporal violations is to reduce, or ideally remove, the delays of workflow execution with the sacrifice of additional cost and overhead. Conventional strategies for handling temporal violations such as resource recruitment and workflow restructure are usually very expensive and/or time consuming [6, 12]. To

avoid these heavy-weight strategies, recoverable violations (compared with “unrecoverable” severe temporal violations which can only be handled by heavy-weight strategies) need to be identified first and then handled by light-weight strategies in a cost-effective fashion.

This paper proposes a novel automatic and cost-effective exception handling framework which consists of three levels of fine-grained temporal violations including level I, level II and level III temporal violations defined within the recoverable probability range, and three corresponding light-weight effective exception handling strategies including TDA (Time Deficit Allocation), ACOWR (Ant Colony Optimisation based two-stage Workflow local Rescheduling) and the TDA+ACOWR (the combined strategy of TDA and ACOWR). Simulation experiments are conducted in our SwinDeW-G scientific grid workflow system [14] to evaluate the effectiveness of our exception handling framework. The experimental results demonstrate the excellent performance of our framework in reducing both local and global temporal violation rates.

The remainder of the paper is organised as follows. Section 2 presents the related work and problem analysis. Section 3 introduces the preliminaries of our work. Section 4 proposes the novel light-weight exception handling framework. Section 5 demonstrates the experimental results. Finally, Section 6 addresses the conclusions and future work.

## 2. Related work and problem analysis

### 2.1 Related work

Temporal constraint is one of the most important workflow QoS constraints. In practice, a set of temporal constraints can be deemed as a QoS contract between clients and service providers [10]. In order to successfully fulfil these QoS contracts, efficient monitoring mechanisms such as checkpoint selection [3] and temporal verification [2] are implemented to dynamically detect temporal violations.

A temporal consistency model is often employed to define temporal violations. The conventional binary temporal consistency model only defines one type of temporal violation, i.e. temporal inconsistency state [17]. However, as proposed in [2], in order to reduce the exception handling cost, the conventional temporal inconsistency state is divided into three levels of fine-grained temporal violations including WC (weak consistency), WI (weak inconsistency) and SI (strong inconsistency). In such a case, fine-grained temporal violations are hoping to be handled by exception handling strategies with different weights, so that the overall exception handling cost could be reduced. Unfortunately, the study on their corresponding exception handling strategies is still in its infancy.

The work in [13] proposes three alternate courses of recovery action being no action (NIL), rollback (RBK) and compensation (COM). NIL, which counts on the automatic recovery of the system itself, is normally not considered ‘risk-free’. As for RBK, unlike handling conventional system function failures, it normally causes extra delays and makes

the current temporal violations even worse. In contrast, COM, namely time deficit compensation, is suitable for handling temporal violations. The work in [2] proposes a time deficit allocation (TDA) strategy which compensates current time deficits by utilising the expected time redundancy of subsequent activities. However, since the time deficit has not been truly reduced by TDA, this strategy can only delay the violations of local constraints on some local workflow segments, but has no effectiveness on global constraints, e.g. the final deadlines. Besides many others, one of the compensation processes which is often employed and can actually make up the time deficit is to amend the schedule of the workflow activities, i.e. workflow rescheduling [9]. Workflow rescheduling is normally triggered by the violation of QoS constraints. Workflow scheduling as well as workflow rescheduling are classical NP-complete problems [8, 11]. Therefore, many heuristic algorithms are proposed. The work in [16] has presented a systematic overview of workflow scheduling algorithms for scientific grid computing. In recent years, many metaheuristic methods such as GA (genetic algorithm) and SA (simulated annealing) have been proposed and exhibit satisfactory performance. Among them, ACO (ant colony optimisation), a type of optimisation algorithm inspired by the foraging behaviour of real ants in the wild, has been adopted to address large complex scheduling problems and proved to be quite effective in many distributed and dynamic resource environments, such as parallel processor systems and grid workflow systems [15, 16]. The work in [4] proposes an ant colony optimisation approach to address scientific workflow scheduling problems with various QoS requirements such as reliability, makespan and cost. The work in [9] introduces an ACO based local rescheduling strategy to handle all types of temporal violations. However, in such a case, it is not cost-effective for minor temporal violations with small delays.

To the best of our knowledge, there is so far no existing work which addresses the problems on both fine-grained temporal violations and their corresponding exception handling strategies in a well-defined exception handling framework.

### 2.2 Problem analysis

To satisfy the two fundamental requirements of *Automation* and *Cost-effectiveness*, the following are the two major issues to be solved.

1) *How to define fine-grained recoverable temporal violations.* In order to avoid those heavy-weight exception handling strategies such as resource recruitment and workflow restructure, fine-grained temporal violations, especially those with tolerable time deficits, are required to be defined upfront. To define temporal violations, we need to employ a temporal consistency model. In recent years, the multiple-state based temporal consistency model which divides traditional inconsistency state into three discrete fine-grained states has been widely applied [2]. However, since this model only utilises static attributes such as the maximum,

mean and minimum activity durations, it lacks the ability to support probability analysis. In complex system environments such as scientific workflows, probability based models are normally much more practical than deterministic models [7]. Therefore, a continuous-state based temporal consistency model where activity durations are modelled as independent random variables is proposed to estimate the probability of meeting given temporal constraints at build time [10]. In this paper, we need to define fine-grained recoverable temporal violations which are within the probability range that occurring time deficits could be statistically compensated by light-weight handling strategies without recruiting new resources.

2) *Which light-weight effective exception handling strategies to be facilitated.* Given the two requirements of *Automation* and *Cost-effectiveness*, conventional heavy-weight handling strategies which involve massive overheads and human interventions are not suitable for those temporal violations with tolerable time deficits. Therefore, to address those fine-grained temporal violations along scientific workflow execution, a set of elegant light-weight effective exception handling strategies need to be employed or designed to automatically handle different levels of temporal violations which are within their capabilities. However, since the capabilities of light-weight handling strategies are relatively limited compared with their heavy-weight counterparts, it is not realistic to replace heavy-weight handling strategies in all situations especially those with extremely severe violations. The objective of this paper is to apply light-weight handling strategies as long as the current temporal violations are within the recoverable probability range.

### 3. Preliminaries

#### 3.1 Probability based temporal consistency model

The probability based run-time temporal consistency model is often applied for statistical analysis on workflow QoS [10]. The model adopts the “3  $\sigma$ ” rule which means with a probability of 99.73% that the sample from a normal distribution model is falling into the interval of  $(\mu - 3\sigma, \mu + 3\sigma)$  [7]. The “3  $\sigma$ ” rule has been widely used in the statistics theory and the value of  $\mu$  and  $\sigma$  can be estimated by the sample mean and sample standard deviation obtained from scientific workflow system logs through statistical methods. Accordingly, the maximum, mean and minimum durations of activity  $a_i$  are defined as  $D(a_i) = \mu_i + 3\sigma_i$ ,  $M(a_i) = \mu_i$  and  $d(a_i) = \mu_i - 3\sigma_i$  respectively where  $\mu_i$  is sample mean and  $\sigma_i$  is the sample standard deviation. Its actual duration at run time is denoted as  $R(a_i)$ . As explained in [10], the workflow completion time can be estimated with the joint normal distribution of individual activity durations. Here, the probability based run-time temporal consistency model is defined.

**Definition: (Probability Based Run-Time Temporal Consistency Model).** At the run-time execution stage, at activity point  $a_p$  where  $p \leq l$  (i.e.  $a_l$  is a subsequent activity of  $a_p$ ), the upper bound constraint  $U(a_k, a_l)$  with the value of  $u(a_k, a_l)$ , where  $k \leq p$  (i.e.  $a_p$  is a subsequent activity of  $a_k$ ), is said to be of:

1) Absolute Consistency (AC),

$$\text{if } R(a_k, a_p) + \sum_{j=p+1}^l (\mu_j + 3\sigma_j) < u(a_k, a_l),$$

2) Absolute Inconsistency (AI),

$$\text{if } R(a_k, a_p) + \sum_{j=p+1}^l (\mu_j - 3\sigma_j) > u(a_k, a_l),$$

3)  $\alpha\%$  Consistency ( $\alpha\%$  C),

$$\text{if } R(a_k, a_p) + \sum_{j=p+1}^l (\mu_j + \lambda\sigma_j) = u(a_k, a_l).$$

Here,  $U(a_k, a_l)$  with the value of  $u(a_k, a_l)$  denotes an upper bound temporal constraint which covers the activities from  $a_k$  to  $a_l$ .  $R(a_k, a_p)$  denotes the sum of run-time durations,  $\lambda$  ( $-3 \leq \lambda \leq 3$ ) is defined as the  $\theta\%$  confidence percentile with the cumulative normal distribution function

$$\text{of } F(\mu_i + \lambda\sigma_i) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\mu_i + \lambda\sigma_i} \frac{e^{-\frac{(x-\mu_i)^2}{2\sigma_i^2}}}{2\sigma_i^2} \cdot dx = \alpha\% \text{ with } 0 < \alpha < 100 [7].$$

#### 3.2 Recoverable temporal violations

In the probability based temporal consistency model, every temporal consistency state is represented with a unique probability value and they together form a continuous Gaussian curve which stands for the cumulative normal distribution [7]. Based on this model, the continuous probability range of (0.13%, 99.87%) is defined as the recoverable probability range where temporal violations can be handled by light-weight exception handling strategies. The reason can be explained as follows. Since the maximum and minimum duration for each activity are defined as  $D(a_i) = \mu_i + 3\sigma_i$  and  $d(a_i) = \mu_i - 3\sigma_i$  respectively, as proved in [10], the overall completion time of the entire workflow instance can be estimated with the normal distribution model and has a statistical lower bound of  $\mu - 3\sigma$  (with 0.13% consistency) and an upper bound of  $\mu + 3\sigma$  (with 99.87% consistency) where  $\mu$  and  $\sigma$  are the joint normal mean and standard deviation respectively for the durations of all activities included. Therefore, all of the probability temporal consistency states which are above the probability of 99.87% are defined as absolute consistency (AC) since they are far from temporal violations and require no actions. In this case, the global temporal constraints allow all activities to be executed with the durations of the mean plus three times standard deviation. Therefore, only with a probability of 1-99.87%, i.e. 0.13%, the global temporal constraints will be violated. Meanwhile, all of the probability temporal

consistency states which are below the probability of 0.13% are defined as absolute inconsistency (AI) since they are severe temporal violations and require heavy-weight handling strategies. In this case, the global temporal constraints only allow all activities to be executed with the durations of the mean minus three times standard deviation. Therefore, with a high probability of 1-0.13%, i.e. 99.87%, the global temporal constraints will be violated. For AI, heavy-weight exception handling strategies are required. In this paper, we focus on the violations within the recoverable range.

#### 4. A novel exception handling framework

##### 4.1 Framework overview

In our framework, as shown in Figure 1, three levels of fine-grained recoverable temporal violations including level I, level II and level III are defined and their corresponding light-weight exception handling strategies are TDA, ACOWR and TDA+ACOWR respectively. In this section, we focus on the definitions for the three levels of temporal violations while the details for the exception handling strategies will be presented later in Section 4.2.

As discussed in Section 3.2, apart from AC and AI, all the probability temporal consistency states within the probability range of (0.13%, 99.87%) may produce temporal violations but statistically can be recovered by light-weight handling strategies. In our framework, the recoverable probability range is further divided into three levels of fine-grained temporal violations.

As can be seen in Figure 1, the dividing points include the normal percentile of  $\lambda_\theta$  and 0 which correspond to the probability consistency states of  $\theta\%$  and 50% respectively. Here,  $\theta\%$  denotes the minimum acceptable initial temporal consistency state and it is usually specified through the negotiation between clients and service providers for setting temporal constraints [10]. In practice,  $\theta\%$  is normally around or above 84.13%, i.e.  $\mu + \sigma$ , which denotes reasonable confidence for on-time completion. Therefore, if current temporal consistency state of  $\alpha\%$  is larger than  $\theta\%$ , the QoS contract still holds but slightly away from absolute consistency, namely, there are still chances for temporal violations. Therefore, for those temporal violations with the

probability consistency states larger than  $\theta\%$ , they are defined as level I temporal violations. As for the second dividing point of 50%, the motivation is the same as in multi-state based temporal consistency model where the dividing point for weak consistency and weak inconsistency is defined with mean durations [2]. Since when the probability consistency state is of 50%, the workflow process needs to be completed within the sum of mean activity durations. Theoretically, in such a situation, the probabilities for temporal violation and non-violation for any distribution models are equally the same, i.e. 50%. Therefore, for those temporal violations with the probability consistency states equal to or larger than 50%, similar to conventional weak consistency states, they are defined as level II temporal violations and a light-weight handling strategy is required to compensate the occurring time deficits and bring the current temporal consistency state back to at least  $\theta\%$ , i.e. the minimum acceptable temporal consistency state. As for those temporal violations with the probability consistency states smaller than 50%, similar to conventional weak inconsistency states, they are defined as level III temporal violations and more efforts compared with that of handling level II temporal violations are required to bring the current temporal consistency state back to at least  $\theta\%$ . However, since they are still within the recoverable probability range rather than absolute inconsistency, level III temporal violations can still be recovered by light-weight automatic handling strategies.

Note that since the probability based temporal consistency model is fully compatible with its multiple-state based counterpart, the state-of-the-art checkpoint selection strategies [3] can be applied directly to detect the three levels of fine-grained temporal violations in our framework. Therefore, in this paper, we only focus on how to handle these temporal violations rather than how to detect them.

##### 4.2 Algorithms for corresponding exception handling strategies

###### 4.2.1 TDA for level I violations

TDA is often used to actively propagate small time deficits so that they could be compensated by the saved execution time of the subsequent workflow activities [2, 10]. In our framework, TDA is responsible for handling level I

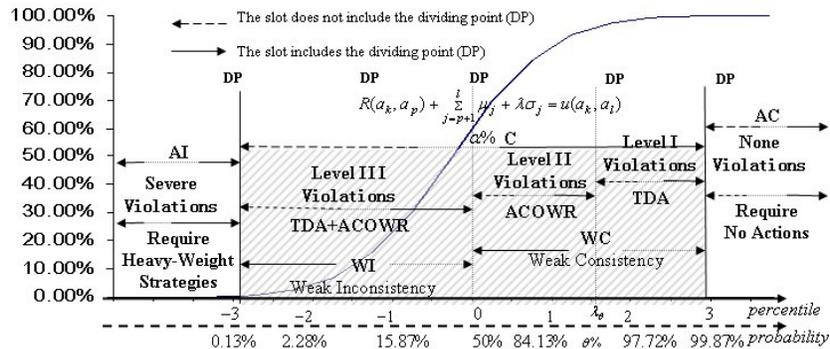


Figure 1. The probability-based run-time temporal consistency model and fine-grained recoverable temporal violations

temporal violations. The pseudo-code for TDA is presented in Figure 2.

---

**Strategy I: TDA**

**Input:** Time deficit detected at activity  $a_p$ ,  $TD(a_p)$ ;  
The next workflow segment  $WS(a_{p+1}, a_{p+1}, \dots, a_{p+m})$ ;  
The temporal constraint  $U(a_{p+1}, a_{p+m})$ ;  
Activity duration models  $M(\mu_i, \sigma_i^2)$ ;

**Output:** Re-assigned temporal constraints  
for each activity  $U'(a_i) | i = p+1, \dots, p+m$ ;

---

//Calculating the expected time redundancy TR  
1)  $TR(a_{p+1}, a_{p+m}) = U(a_{p+1}, a_{p+m}) - \sum_{i=p+1}^{p+m} (\mu_i + \lambda_{\sigma} * \sigma_i)$  ;  
// allocating the time deficit to the subsequent activities based on their mean activity time redundancy  
2) for  $i = p+1$  to  $p+m$   
3)  $U'(a_i) = U(a_i) - TR(a_{p+1}, a_{p+m}) * \frac{D(a_i) - M(a_i)}{\sum_{i=p+1}^{p+m} (D(a_i) - M(a_i))}$

//based on  $3\sigma$  rule,  $D(a_i) = \mu_i + 3\sigma_i$  and  $M(a_i) = \mu_i$   

$$= U(a_i) - TR(a_{p+1}, a_{p+m}) * \frac{\sigma_i}{\sum_{i=p+1}^{p+m} (\sigma_i)}$$

4) end for  
5) return  $U'(a_i) | i = p+1, \dots, p+m$

---

Figure 2. Algorithm for TDA

The first task of TDA is to calculate the expected time redundancy of the next workflow segment (Line 1). Normally, the next workflow segment is chosen as the workflow segment between the next activity of the checkpoint  $a_p$ , i.e.  $a_{p+1}$ , and the end activity of the next local temporal constraint, say  $a_{p+m}$  here. The expected time redundancy is defined as the difference between the temporal constraint and the execution time for the minimum acceptable temporal consistency state (Line 1). The expected time redundancy can be borrowed to compensate the time deficits for level I temporal violations. After that, the expected time redundancy is allocated to the subsequent activities within the workflow segment according to the proportion of their mean activity time redundancy as defined in [2] (Line 2 to Line 4). After that, re-assigned temporal constraints for each activity are returned (Line 5). Therefore, the actual compensation process for TDA is to tighten the temporal constraints of the subsequent activities so as to ensure that the current scientific workflow execution is close to absolute consistency. However, since TDA does not decrease the actual activity durations of the subsequent activities, it can handle level I violations of some local workflow segments but has no effectiveness on global constraints, e.g. the final deadline. To actually decrease the execution time of workflow segments (required by level II and level III temporal violations), more sophisticated handling strategies such as workflow rescheduling are required.

#### 4.2.2 ACOWR for level II violations

ACOWR [9] is responsible for handling level II temporal violations. ACOWR stands for Ant Colony Optimisation based two-stage Workflow local Rescheduling strategy. Here, “two-stage” means a two-stage searching process designed in our algorithm to strike a balance between time deficit compensation and the completion time of other activities while “local” means the rescheduling of “local” workflow

segments with “local” resources in the integrated task-resource list [9]. Since ACOWR has been presented in [9], its technical details are omitted in this paper. Here, we mainly illustrate its two searching stages. The pseudo-code for ACOWR is presented in Figure 3.

---

**Strategy II: ACOWR**

**Input:** Time deficit detected at activity  $a_p$ ,  $TD(a_p)$ ;  
Integrated task-resource list  
 $L(a_i, R_i) | i = p+1, \dots, p+n, j = 1, 2, \dots, K$  ;  
DAG task graphs  $DAG(\sigma_i | a_j \leq a_m)$ ;  
Activity duration models  $M(\mu_i, \sigma_i^2)$ ;  
Resource peers  $R_i, ES(R_i), Cost(R_i) | i = 1, 2, \dots, K$ .

**Output:** Rescheduled task-resource list

---

//Stage 1: Optimising the overall execution time and cost for the integrated task-resource list through ACO algorithm  
1) While (stopping condition is not met)  
{  
// ACO based optimisation on total makespan and total cost  
2) ACO();  
// return the best-so-far solution and record into the SolutionSet  
3) Return(Solution, SolutionSet)  
}  
//Stage 2: Searching for the BestSolution from SolutionSet  
4) While (not end of the SolutionSet)  
{  
// compare the compensation time of each Solution with the time deficit, discard the Solution if it is smaller than the time deficit  
5) COMPARE(Solution.ct, TD(a<sub>p</sub>));  
// compare all remaining Solution and set BestSolution as the one with the minimum cost  
6) BestSolution = MIN(Solution.cost);  
}  
7) Return the BestSolution;  
// return the rescheduled integrated task-resource list and deploy  
8) DEPLOY(L)

---

Figure 3. Algorithm for ACOWR

In the first searching stage, the ACO algorithm is applied to optimise the total makespan and total cost of the integrated task-resource list which consists of all the individual tasks lists on the local resources (Line 1 to Line 3). During this stage, a *SolutionSet* is being created where the best solution (with the largest fitness value which is defined as a reciprocal to the total makespan and total cost) for each iteration is stored. In the second searching stage (Line 4 to Line 6), the *BestSolution* which can compensate the time deficit detected at the checkpoint is retrieved from the *SolutionSet*. The compensation time of each solution (i.e. the difference between the scheduled completion time of the violation workflow segment before and after rescheduling) is first compared with the time deficit to filter out unsuccessful solutions. Here, the *BestSolution* is defined as the one with the minimum overall cost among all the successful solutions. Finally, the *BestSolution*, i.e. the solution which can handle the occurring temporal violations while having the minimum overall cost is returned (Line 7) as the regenerated scheduling plan for the integrated task-resource list and ready to be deployed (Line 8).

#### 4.2.3 TDA+ACOWR for level III violations

The combined strategy of TDA and ACOWR (denoted as TDA+ACOWR) is responsible for handling level III temporal violations. ACOWR is capable of removing most time deficits and handle level II temporal violations. However, due to the larger amount of time deficits occurring in level III temporal violations, we propose the combined

strategy of TDA and ACOWR to achieve a stronger exception handling capability. The pseudo-code for TDA+ACOWR is presented in Figure 4.

```

Strategy III: TDA+ACOWR


---


Input: Time deficit detected at activity  $a_p$   $TD(a_p)$ ;
The next workflow segment  $WS(a_{p+1}, a_{p+1}, \dots, a_{p+m})$ ;
Integrated task-resource list
 $L\{a_i, R_j\} | i = p+1, \dots, p+n, j = 1, 2, \dots, K$ ;
DAG task graphs  $DAG\{G_i | a_j \leq a_m\}$ ;
The temporal constraint  $U(a_{p+1}, a_{p+m})$ ;
Activity duration models  $M\{\mu_j, \sigma_j^2\}$ ;
Resource peers  $R\{r_i, ES(r_i), Cost(r_i) | i = 1, 2, \dots, K\}$ .
Output: Rescheduled task-resource list
// Stage 1: Try to adjust level III temporal violation back to level II by TDA
// decrease the current time deficit by the expected time redundancy of the next
// workflow segment but without allocating the time deficit
1)  $TR(a_{p+1}, a_{p+m}) = U(a_{p+1}, a_{p+m}) - \sum_{i=p+1}^{p+m} (\mu_i + \lambda_{a_i} * \sigma_i)$ ;
2)  $TD(a_p) = TD(a_p) - TR(a_{p+1}, a_{p+m})$ ;
// Stage 2: Compensate the remaining time deficit by ACOWR
3) While ( $TD(a_p) > 0$  or the stopping condition is not met)
{
// call Strategy II
4) ACOWR();
// decrease the time deficit by the compensation time of the BestSolution
5)  $TD(a_p) = TD(a_p) - BestSolution$ ;
6) if  $TD(a_p) > 0$ 
// read in the second next workflow segment
{
7)  $WS = WS(a_{p+m+1}, a_{p+m+2}, \dots, a_{p+m+n})$ ;
// update all the required information
8)  $UPDATE(L, DAG, M, R)$ ;
}
}
// return the rescheduled integrated task-resource list and deploy
9) DEPLOY(L)

```

Figure 4. Algorithm for TDA+ACOWR

The combined strategy starts with the first stage of TDA (Line 1 to Line 2). However, here we only utilise the expected time redundancy to decrease the current time deficits without allocating them since the subsequent activities will be further rescheduled by ACOWR. The second stage is an iterative process of ACOWR (Line 3 to Line 8). Here, ACOWR is called for the first time to compensate the time deficit with the best solution for the next workflow segment (Line 4 to Line 5). However, if the time deficit is not removed, the second next workflow segment is read in to increase the size of local workflow segment and the input information is updated accordingly (Line 7 to Line 8). Afterwards, as the iteration process carries on, ACOWR will optimise additional workflow segments until the time deficit is entirely removed. In practice, according to our experimental results as will be demonstrated in Section 6, two consecutive workflow segments are usually more than enough to compensate the occurring time deficits. Therefore, ACOWR will normally be applied no more than twice in the TDA+ACOWR strategy for handling level III temporal violations. Finally, the optimised integrated task-resource list will be returned and deployed (Line 9).

## 5. Evaluation

In this section, we evaluate the effectiveness of our exception handling framework. In our framework, three light-weight effective exception handling strategies are facilitated for handling three levels of fine-grained recoverable temporal violations. First, these three strategies can all be implemented automatically without human

intervention; Second, these three strategies are all light-weight since no additional resources are required to be recruited at workflow run time. Therefore, in a qualitative fashion, we can claim that our framework satisfies the fundamental requirements of *Automation* and *Cost-effectiveness*.

Next, in a quantitative fashion, the performance of ACOWR is first evaluated and then followed by the simulation experiments on the evaluation of the framework. Due to the space limit, more comprehensive experimental results and all the program codes can be found online<sup>1</sup>.

### 5.1. Simulation environment and experiment settings

SwinDeW-G (Swinburne Decentralised Workflow for Grid) is a peer-to-peer based scientific grid workflow system running on the SwinGrid (Swinburne service Grid) platform [14]. SwinGrid contains many grid nodes distributed in different places. 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) Node, Swinburne ESR (Enterprise Systems Research laboratory) Node, Swinburne Astrophysics Supercomputer Node, and Beihang CROWN (China R&D environment Over Wide-area Network) Node in China. They are running Linux, GT4 (Globus Toolkit) or CROWN grid toolkit 2.5 where CROWN is an extension of GT4 with more middleware, hence compatible with GT4. In SwinDeW-G, a scientific workflow is executed by different peers that may be distributed at different grid nodes.

The real world examples we investigated for this paper include a pulsar searching scientific workflow in Astrophysics and a weather forecast scientific workflow in Meteorology. The detailed introduction and the experiment settings are included in the online document due to the space limit. Note that the experiment settings are based on the statistics of the system logs but generalised deliberately with larger ranges and variances in order to represent a more general system environment.

### 5.2. Simulation experiment and results

First, we demonstrate the experimental results on the performance ACOWR which is the core component in our framework. The capability of ACOWR in the handling of temporal violations can be quantitatively measured by its compensation ratio defined as follows.

$$CompensationRatio = 100\% - \frac{LocalMakespanAfter}{LocalMakespanBefore}$$

where the *LocalMakespanAfter* is the execution time of the activities in the local workflow segment of the violated workflow instance, after workflow rescheduling; and *LocalMakespanBefore* is the corresponding execution time before workflow rescheduling. Figure 5 depicts the

<sup>1</sup><http://www.ict.swin.edu.au/personal/xliu/doc/HandlingFramework.rar>

compensation ratio for each round of experiment where the total number of workflow segment activities increases from 50 to 300 and the total number of resources increases from 3 to 20. For more detailed settings, please see the online document. For ACOWR, the maximum, minimum and mean compensation ratios are 69.0%, 35.0% and 58.5% respectively. It can be seen that the compensation ratio is on a roughly increasing trend with the growing number of activities. Meanwhile, the curve fluctuates around a stable value (e.g. 62.5%) when the number of activities is larger enough, i.e. over 200. Therefore, it can be foreseen that the performance of ACOWR tends to become more stable when the size of the integrated task-resource list increases. The results demonstrate that ACOWR can compensate the time deficit by effectively reducing the local workflow makespan of the violated workflow segment.

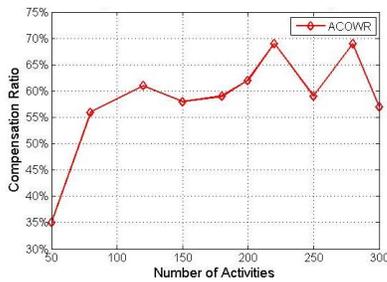


Figure 5. Compensation on Violated Workflow Segment

Note that we have also evaluated the performance of ACOWR with more measurements such as the optimisation ratio on total makespan and total cost, and compared with that of GA based workflow rescheduling strategy where their results on the *CompensationRatio* are very close to each other. Specifically, the mean optimisation ratio on total makespan is around 15.9% and the mean optimisation ratio on total cost is around 13.4%. Meanwhile, the average CPU time for running ACOWR on a SwinDeW-G node is around 4.73 seconds which can be regarded as negligible compared with the durations of scientific workflow activities usually measured in minutes and hours. Due to the space limit, we

only focus on the handling capability of ACOWR in this paper while omitting other details which could all be found in our online document.

After the evaluation of ACOWR, we further implement our exception handling framework into various sizes of scientific workflows to verify its performance in the handling of temporal violations. In our experiment, the size of scientific workflows ranges from 500 to 20,000. For every group of 20 to 50 activities, an upper bound temporal constraint is assigned. The strategy for setting temporal constraint is adopted from the work in [10] where a normal percentile is used to specify temporal constraints and denotes the expected probability for on-time completion. Here, we conduct three rounds of independent experiments where the temporal constraints are set with different normal percentiles of 1.00, 1.15 and 1.28 which denotes the probability of 84.1%, 87.5% and 90.0% for on-time completion if without any handling strategies on temporal violations (denoted as COM(1.00), COM(1.15) and COM(1.28)). For the comparison purpose, we record the local and global violation rates under natural situations, i.e. without any handling strategies (denoted as NIL), and compared with that of standalone TDA strategy and our framework (denoted as Framework). Here, for fairness, if and only if the time deficits can be compensated within the next group of activities, the previous local constraint is not considered as violated. Otherwise, one local violation is counted. On the contrary, the global temporal violations are decided by the actual completion time of the entire workflow and the global temporal constraints. Each experiment is executed for 100 times to get the average violation rates.

As can be seen in Figure 6, in the subplots on the left hand side, the violation rates behave stably under different constraint settings. For example, the average local violation rate is around 15% in COM(1.00) where the probability for on-time completion is 84.1%. As for TDA, it can reduce the local violation rates when the size of scientific workflow is small. However, it behaves poorly when the number of activities exceeds 4,000. As analysed in Section 4.2.1, since TDA does not compensate time deficits, it cannot postpone

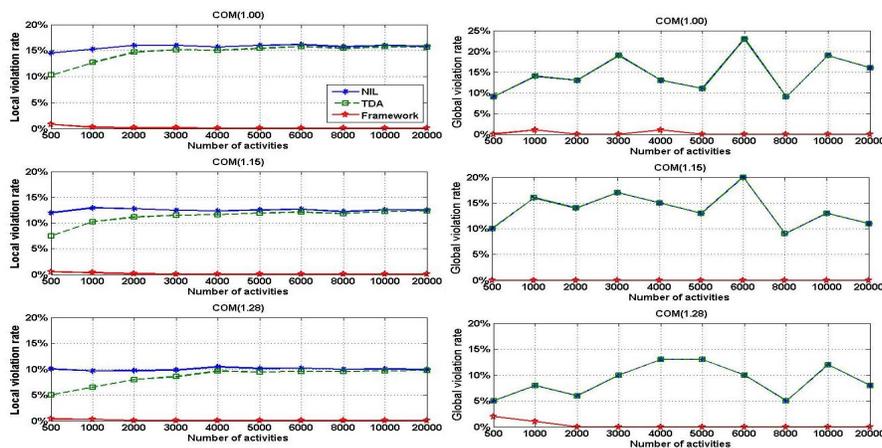


Figure 6. Effectiveness of handling temporal violations

temporal violations any more when the time deficits accumulates to become large enough. With our handling framework, local violation rate is kept close to zero since the three handling strategies, TDA, ACOWR and TDA+ACOWR, can be applied dynamically to tackle different levels of temporal violations. The average local violation rates for our framework in each round are 0.16%, 0.13% and 0.09% respectively. The subplots on the right hand side show the results on global violation rates. Since TDA has no effectiveness on global violations, the global violation rates for NIL and TDA are overlapped. The global violation rates for NIL and TDA behave very unstably but increase roughly with the number of activities while decreasing with the value of normal percentiles. The average global violation rates for NIL and TDA in each round are 14.6%, 13.8% and 9.0% respectively. With our handling framework, the global violation rate is kept close to zero since most local temporal violations are handled automatically along workflow executions. The average global violation rates of our framework in each round are 0.2%, 0.0% and 0.3% respectively.

To conclude, the experimental results show that our exception handling framework has excellent performance in the handling of different levels of fine-grained recoverable temporal violations and thus can significantly reduce both local and global temporal violation rates in scientific workflows.

## 6. Conclusions and Future Work

In this paper, the problem of handling temporal violations in scientific workflows has been investigated and addressed by our proposed novel exception handling framework. Given the two fundamental requirements of *Automation* and *Cost-effectiveness*, our framework consists of three levels of fine-grained temporal violations within the recoverable probability range including level I, level II and level III, and their corresponding light-weight effective handling strategies including TDA, ACOWR and TDA+ACOWR. The experiments conducted in our SwinDeW-G scientific grid workflow system have demonstrated the excellent performance of ACOWR in compensating the occurring time deficit. Furthermore, with scientific workflows of various sizes and different temporal constraints settings, our framework has been verified with significant improvements over NIL and standalone TDA in reducing both local and global temporal violation rates.

In the future, some other scheduling algorithms such as Min-Min, GA and SA will be adopted and compared with ACOWR, and some heavy-weight strategies will be further investigated and improved to handle severe temporal violations in scientific workflows.

## ACKNOWLEDGMENTS

This work is partially supported by Australian Research Council under Linkage Project LP0990393, the National Natural Science Foundation of China project No. 70871033.

## REFERENCES

- [1] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," *Future Generation Computer Systems*, vol. 25, no. 6, pp. 599-616, 2009.
- [2] J. Chen and Y. Yang, "Multiple States based Temporal Consistency for Dynamic Verification of Fixed-time Constraints in Grid Workflow Systems," *Concurrency and Computation: Practice and Experience*, Wiley, vol. 19, no. 7, pp. 965-982, 2007.
- [3] J. Chen and Y. Yang, "Temporal Dependency based Checkpoint Selection for Dynamic Verification of Temporal Constraints in Scientific Workflow Systems," *ACM Transactions on Software Engineering and Methodology*, available at <http://www.swinflow.org/papers/TOSEM.pdf>, accessed on 1<sup>st</sup> Jun 2010.
- [4] W. Chen and 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, pp. 29-43, 2009.
- [5] E. Deelman, D. Gannon, M. Shields, and I. Taylor, "Workflows and e-Science: An Overview of Workflow System Features and Capabilities," *Future Generation Computer Systems*, vol. In Press, Corrected Proof, no.
- [6] C. Hagen and G. Alonso, "Exception Handling in Workflow Management Systems," *IEEE Trans. on Software Engineering*, vol. 26, no. 10, pp. 943-958, 2000.
- [7] A. M. Law and W. D. Kelton, *Simulation Modelling and Analysis (Fourth Edition)*: McGraw-Hill, 2007.
- [8] C. Liu and S. Baskiyar, "Scheduling Mixed Tasks with Deadlines in Grids Using Bin Packing", *Proc. 14th IEEE International Conference on Parallel and Distributed Systems (ICPADS '08)*, pp. 229-236, 2008.
- [9] X. Liu, J. Chen, Z. Wu, Z. Ni, D. Yuan, and Y. Yang, "Handling Recoverable Temporal Violations in Scientific Workflow Systems: A Workflow Rescheduling Based Strategy", *Proc. 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid10)*, pp. 534-537, Melbourne, Australia, 2010.
- [10] X. Liu, Z. Ni, J. Chen, and Y. Yang, "A Probabilistic Strategy for Temporal Constraint Management in Scientific Workflow Systems," *Concurrency and Computation: Practice and Experience*, available at <http://www.ict.swin.edu.au/personal/xliu/doc/ConstraintManagement.pdf>, accessed on 1<sup>st</sup> Jun 2010.
- [11] Y. Ma and B. Gong, "Double-layer Scheduling Strategy of Load Balancing in Scientific Workflow", *Proc. 15th International Conference on Parallel and Distributed Systems (ICPADS'09)*, pp. 671-678, 2009.
- [12] R. Prodan and T. Fahringer, "Overhead Analysis of Scientific Workflows in Grid Environments," *IEEE Trans. on Parallel and Distributed Systems*, vol. 19, no. 3, pp. 378-393, 2008.
- [13] N. Russell, W. M. P. van der Aalst, and A. H. M. ter Hofstede, "Workflow Exception Patterns", *Proc. 18th International Conference on Advanced Information Systems Engineering (CAiSE'06)*, Lecture Notes in Computer Science, vol. 4001, pp. 288-302, Berlin, Germany, 2006.
- [14] Y. Yang, K. Liu, J. Chen, J. Lignier, and H. Jin, "Peer-to-Peer Based Grid Workflow Runtime Environment of SwinDeW-G", *Proc. 3rd International Conference on e-Science and Grid Computing (e-Science07)*, pp. 51-58, Bangalore, India, Dec. 2007.
- [15] J. Yu and R. Buyya, "Scheduling Scientific Workflow Applications with Deadline and Budget Constraints Using Genetic Algorithms," *Scientific Programming*, vol. 14, no. 3,4, pp. 217-230, Dec. 2006.
- [16] J. Yu and R. Buyya, "Workflow Scheduling Algorithms for Grid Computing," Technical Report GRIDS-TR-2007-10, Computing and Distributed Systems Laboratory, The University of Melbourne, Australia, May 31, 2007.
- [17] H. Zhuge, T. Cheung, and H. Pung, "A Timed Workflow Process Model, *Journal of Systems and Software*," vol. 55, no. 3, pp. 231-243, 2001.