

# A novel general framework for automatic and cost-effective handling of recoverable temporal violations in scientific workflow systems

Xiao Liu<sup>a,\*</sup>, Zhiwei Ni<sup>b</sup>, Zhangjun Wu<sup>b,a</sup>, Dong Yuan<sup>a</sup>, Jinjun Chen<sup>a</sup>, Yun Yang<sup>a</sup>

<sup>a</sup> Faculty of Information and Communication Technologies, Swinburne University of Technology, Hawthorn, Melbourne 3122, Australia

<sup>b</sup> Institute of Intelligent Management, School of Management, Hefei University of Technology, Hefei, Anhui 230009, China

## ARTICLE INFO

### Article history:

Received 14 June 2010

Received in revised form 20 October 2010

Accepted 20 October 2010

Available online 30 October 2010

### Keywords:

Temporal violation

Exception handling

Scientific workflow

Workflow rescheduling

Workflow QoS

Temporal verification

## ABSTRACT

Due to the complex nature of scientific workflow environments, temporal violations often take place and may severely reduce the timeliness of the execution's results. To handle temporal violations in an automatic and cost-effective fashion, two interdependent fundamental issues viz. the definition of fine-grained recoverable temporal violations and the design of light-weight effective exception handling strategies need to be resolved. However, most existing works study them separately without defining a comprehensive framework. To address such a problem, with the probability based temporal consistency model which defines the range of recoverable temporal violations, a novel general automatic and cost-effective exception handling framework is proposed in this paper where fine-grained temporal violations are defined based on the empirical function for the capability lower bounds of the exception handling strategies. To serve as a representative case study, a concrete example exception handling framework which consists of three levels of fine-grained temporal violations and their corresponding exception handling strategies is presented. The effectiveness of the example framework is evaluated by large scale simulation experiments conducted in the SwinDeW-G scientific grid workflow system. The experimental results demonstrate that the example framework can significantly reduce the overall average violation rates of local temporal constraints and global temporal constraints to 0.127% and 0.167% respectively.

© 2010 Elsevier Inc. All rights reserved.

## 1. Introduction

Scientific workflow systems are a type of workflow management systems aiming at supporting complex scientific processes in many e-science applications such as climate modelling, disaster recovery simulation, astrophysics and high energy physics (Deelman et al., 2008; Taylor et al., 2007). Scientific workflow systems can also be seen as a type of high-level middleware services for high performance computing infrastructures such as cluster, grid, peer-to-peer (p2p) or cloud computing (Buyya et al., 2009; Foster and Kesselman, 2004; Kim et al., 2007; Yang et al., 2007). In recent years, due to the growing demand for high performance computing infrastructures and large scale distributed and collaborative e-science applications, scientific workflow systems have been attracting increasing interests from distributed and parallel system researchers in the area of High Performance Computing (HPGC, 2009; PDSEC, 2009) and software engineering researchers

in the area of Software Engineering for Computational Science and Engineering (Chen and Yang, in press; SECES, 2008). One of the common research issues is how to deliver satisfactory workflow QoS (quality of service), i.e. how to satisfy workflow QoS constraints such as the constraints on time, cost, fidelity, reliability and security (Son and Kim, 2001; Yu and Buyya, 2005). Among them, time is one of the basic measurements for system and software performance and hence attracts many researchers in the workflow area (van der Aalst et al., 2000; Chen and Yang, 2008; Duan et al., 2009; Eder et al., 1999; Li et al., 2004; Yu and Buyya, 2005; Zhuge et al., 2001).

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 (Li et al., 2004; Zeng et al., 2008). 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 programme everyday at, for instance, 6:00 pm. Meanwhile, given the large number of data and computation intensive activities for scientific investigation purposes, scientific workflows are usually deployed on distributed

\* Corresponding author. Tel.: +61 03 92148699.

E-mail addresses: [xliu@swin.edu.au](mailto:xliu@swin.edu.au) (X. Liu), [nzwdg@hfut.edu.cn](mailto:nzwdg@hfut.edu.cn) (Z. Ni), [wuzhangjun@hfut.edu.cn](mailto:wuzhangjun@hfut.edu.cn) (Z. Wu), [dyuan@swin.edu.au](mailto:dyuan@swin.edu.au) (D. Yuan), [jchen@swin.edu.au](mailto:jchen@swin.edu.au) (J. Chen), [yyang@swin.edu.au](mailto:yyang@swin.edu.au) (Y. Yang).

high performance infrastructures such as grid and cloud. Therefore, to deliver satisfactory temporal QoS, the violations of both local temporal constraints (or local violations for short) and global temporal constraints (or global violations for short), need to be proactively detected and handled (Zhuge et al., 2001). Recent studies on temporal verification in scientific workflows mainly focus on runtime checkpoint selection (Chen and Yang, in press) and multiple-state based temporal verification (Chen and Yang, 2007) which can deal with the monitoring of temporal consistency states and the detection of potential temporal violations. However, a significant follow-up issue is how to handle those temporal violations. Till date, work on such an issue is still in its infancy. However, it must be properly addressed so as to guarantee high success rates for on-time completion of scientific workflows. Specifically, two fundamental requirements for handling temporal violations, *automation* and *cost-effectiveness*, need to be considered.

- (1) *Automation*. Due to the complex nature of scientific applications and their distributed running environments such as grid and cloud, a large number of temporal violations may often be expected in scientific workflows. Besides, scientific workflow systems are designed to be highly automatic to conduct large scale scientific processes, human interventions which are normally of low efficiency should be avoided as much as possible, especially during workflow runtime (Deelman et al., 2008). Therefore, similar to dynamic checkpoint selection and temporal verification strategies (Chen and Yang, in press), handling strategies are required to automatically tackle a large number of temporal violations and relieve users from the heavy workload of handling those exceptions.
- (2) *Cost-effectiveness*. The purpose of handling temporal violations is to reduce, or ideally remove, the delays of workflow execution by exception handling strategies with the sacrifice of additional cost which consists of both monetary cost and time overheads. Conventional exception handling strategies for temporal violations, such as resource recruitment and workflow restructure, are usually very expensive (Buhr and Mok, 2000; Hagen and Alonso, 2000; Prodan and Fahringer, 2008; Russell et al., 2006a). The cost for recruiting new resources (e.g. the cost for service discovery and deployment, the cost for data storage and transfer) is normally very large during workflow runtime in distributed computing environments (Prodan and Fahringer, 2008). As for workflow restructure, it is usually realised by the amendment of local workflow segments or temporal QoS contracts, i.e. modifying scientific workflow specifications by human decision makers (Liu et al., 2008b). However, due to budget (i.e. monetary cost) limits and temporal constraints, these heavy-weight strategies (with large monetary cost and/or time overheads) are usually too costly to be practical. To avoid these heavy-weight strategies, recoverable violations (in comparison to severe temporal violations which can be regarded as non-recoverable in practice) need to be identified first and then handled by light-weight strategies (with small monetary cost and/or time overheads) in a cost-effective fashion.

Given the requirement of *Automation*, exception handling strategies need to be designed to handle temporal violations in an automatic fashion without human interventions. Meanwhile, since most strategies have their limits in the capability of recovering temporal violations, different handling strategies are normally only effective for a range of temporal violations with limited amount of time deficits (the time delays given specific temporal constraints). Given the requirement of *Cost-effectiveness*, for all the candidate strategies which are capable of handling the current temporal violation, ideally, only the one with the lowest cost should be applied. Therefore, the definition of fine-grained temporal violations and

the design of exception handling strategies should be investigated as two interdependent tasks within the same exception handling framework. However, since recent studies in temporal verification mainly focus on the detection of temporal violations, fine-grained temporal violations are usually defined for the general application purpose ignoring the performance of exception handling strategies in the specific workflow systems. For example, the work in Chen and Yang (2007) proposes a multiple-states based temporal consistency model. Besides SC (strong consistency) which requires no action, three types of fine-grained temporal inconsistency states including WC (weak consistency), WI (weak inconsistency) and SI (strong inconsistency) are defined based on the minimum, mean and maximum workflow execution time. However, without the investigation on the performance of different exception handling strategies, it is difficult to determine which strategy should be applied to handle the detected temporal violations. Therefore, it is more reasonable that fine-grained temporal violations should be defined according to the selection of different exception handling strategies with different capabilities, rather than most of the previous studies where fine-grained temporal violations are defined in the first place then looking for available exception handling strategies. To the best of our knowledge, this is the first work to systematically investigate a general exception handling framework for automatic and cost-effective handling of temporal violations in scientific workflow systems.

In this paper, along with the probability based temporal consistency model which defines the range of recoverable temporal violations, a novel general automatic and cost-effective exception handling framework is proposed. Specifically, fine-grained temporal violations are first defined based on the empirical function for the capability lower bounds of the exception handling strategies. Afterwards, to serve as a case study, a concrete example framework is presented which consists of three levels of fine-grained temporal violations, viz., level I, level II and level III temporal violations defined within the recoverable probability range, and three light-weight automatic exception handling strategies, viz., TDA (Time Deficit Allocation), ACOWR (Ant Colony Optimisation based two-stage Workflow local Rescheduling) and TDA + ACOWR (the combined strategy of TDA and ACOWR). Large scale simulation experiments are conducted in the SwinDeW-G scientific grid workflow system (Yang et al., 2007) to evaluate the effectiveness of the example framework.

The remainder of the paper is organised as follows. Section 2 presents a motivating example and the problem analysis. Section 3 proposes a general exception handling framework for temporal violations. Section 4 presents a case study with a concrete exception handling framework with three levels of temporal violations and their corresponding handling strategies. Section 5 demonstrates comprehensive simulation results. Section 6 reviews the related work. Finally, Section 7 concludes the paper and points out the future work.

## 2. Motivating example and problem analysis

### 2.1. Motivating example

In this section, we present an example scientific workflow in Astrophysics. Parkes Radio Telescope (<http://www.parkes.atnf.csiro.au/>, located 380 km west of Sydney, Australia), one of the most famous radio telescopes, is serving institutions around the world. Swinburne Astrophysics group has been conducting a pulsar searching survey based on the observation data from Parkes Radio Telescope (<http://astronomy.swin.edu.au/pulsar/>). The pulsar searching process is a typical scientific workflow which involves a large number

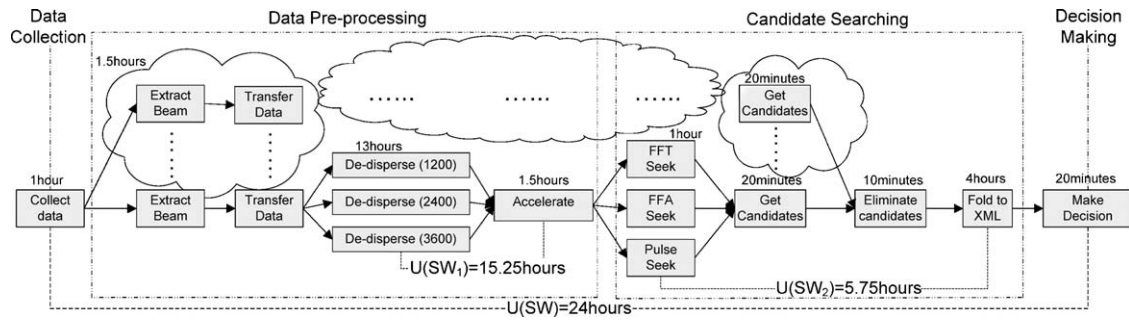


Fig. 1. An example scientific workflow for pulsar searching in astrophysics.

of data intensive and computation intensive activities (see Fig. 1 for details). For a single searching process, the average overall data volume (not including the raw stream data from the telescope) is over 4 terabytes and the average overall execution time is about 23 h on Swinburne high performance supercomputing facility (<http://astronomy.swinburne.edu.au/supercomputing/>).

As depicted in Fig. 1, we illustrate the high-level workflow structure and focus on one of the total 13 parallel paths for different beams. The other parallel paths in the same structure are omitted here and denoted with cloud symbols. The mean durations (normally with large variances) for high-level activities (with sub-processes underneath) and three upper bound temporal constraints are also presented. An upper bound constraint between two activities is a relative time value so that the duration between them must be less than or equal to it (Chen and Yang, in press). The details of these activities could be found on the aforementioned websites.

The entire pulsar searching workflow which includes four major segments (data collection, data pre-processing, candidate searching and decision making) is normally required to be completed in one day, i.e. a global temporal constraint of 24 h, denoted as  $U(SW)$  in Fig. 1. Meanwhile, for the fine-grained control over the workflow execution, e.g. two local temporal constraints on workflow segments, denoted as  $U(SW_1)$  and  $U(SW_2)$  are also assigned based on the probabilistic setting strategy presented in Liu et al. (2008b).  $U(SW_1)$  covers the data pre-processing segment which includes *De-dispersion* and *Accelerate*. A large number of de-dispersion files need to be generated according to different choices of trial dispersion and normally take 13 h for the minimum of 1200 de-dispersion files. *Accelerate* is for binary pulsar searching where every de-dispersion file further generates several accelerated de-dispersion files. The whole process takes around 1.5 h.  $U(SW_2)$  covers the candidate seeking segment which includes searching candidates (with different seeking algorithms such as *FFT seek*, *FFA seek*, and *Pulse seek*), *Get candidates*, *Eliminate candidates* and *Fold to XML* which folds the original beam files of valid pulsar candidates to XML files and visualises them to support the decision of human experts on whether a pulsar has been found or not.

Here, we demonstrate two example scenarios to illustrate the issue of handling temporal violations. For the first example, if *De-dispersion* takes 14.75 h (i.e. a delay of around 15% of its mean), a delay of 1 h will probably occur and thus violates  $U(SW_1)$ . In such a case, the duration of *Accelerate* need to be decreased to at most 0.5 h, i.e. a reduction about 67% of its mean duration, in order to handle the violation of  $U(SW_1)$ . For the second example, if the three activities before *Fold to XML* take a total of 1.8 h (a delay of around 20% of the mean), there will probably occur a delay of 3 min. In such a case, the duration of *Fold to XML* needs to be decreased by 3 min, i.e. a reduction of 1.25% of its mean duration, in order to handle the violation of  $U(SW_2)$ . Upon our observation, time delays, either major delays like the first scenario or minor delays like the sec-

ond one, may often occur along scientific workflow execution and result in frequent temporal violations which deteriorate the overall temporal QoS in scientific workflow systems. However, it is not a trivial issue to decrease activity durations during workflow runtime since the resource competition is normally very severe. For example, in Swinburne high performance supercomputing facility, the average utilisation ratio for high performance computing units is around 80% for most of the time according to the system's historic information monitored by the Swinburne Supercomputer Group. In practice, workflow rescheduling is often employed to address the violations of QoS constraints (Yu and Shi, 2007). However, additional cost in budget and time is required for workflow rescheduling. In the above two scenarios, the violation of  $U(SW_1)$  normally requires much more efforts (a reduction of 67%) to be handled compared with the violation of  $U(SW_2)$  (a reduction of 1.25%). However, are they, especially the first one, still recoverable by light-weight exception handling strategies without recruiting additional resources outside of the current system? Are there any exception handling strategies which can recover both of them? To handle temporal violations in scientific workflow systems, we need to answer these questions.

## 2.2. Problem analysis

As mentioned earlier, the two fundamental requirements for handling temporal violations in scientific workflows are *automation* and *cost-effectiveness*. To meet these two requirements, there are two major issues which need to be solved: *how to define fine-grained recoverable temporal violations*; and *which light-weight effective exception handling strategies to be facilitated*.

(1) *How to define fine-grained recoverable temporal violations*. In order to avoid 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 (Chen and Yang, 2007). 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 (Law and Kelton, 2007). 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 (Liu et al., 2008b). 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.

- (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 facilitated. 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 rational objective here is to apply light-weight handling strategies as long as the current temporal violations are within the recoverable probability range. Therefore, a set of light-weight effective exception handling strategies with different capabilities need to be employed or designed to tackle various fine-grained temporal violations.

Furthermore, in order to achieve the requirements of *Automation* and *Cost-effectiveness*, the above two problems should be investigated within the same exception handling framework so that fine-grained temporal violations can be defined based on the performance of the handling strategies. Meanwhile, since different scientific workflow systems often employ different exception handling strategies, a general exception handling framework should be designed so that it can be built upon the existing system functionalities rather than from scratch.

### 3. A general exception handling framework for temporal violations

In this section, an overview of a probability based temporal consistency model is presented and the range of recoverable temporal violations is defined. Afterwards, a general exception handling framework is proposed where fine-grained temporal violations are defined based on the empirical function for the capability lower bounds of exception handling strategies.

#### 3.1. Preliminaries

##### 3.1.1. A probability based temporal consistency model

Here, we present an overview of the probability based runtime temporal consistency model to define fine-grained temporal violations (Liu et al., 2008b). For statistical analysis, the “3 $\sigma$ ” rule is adopted which means with a probability of 99.73% that the sample from a normal distribution model is falling into the interval of  $(\mu - 3\sigma, \mu + 3\sigma)$  (Stroud, 2007). The “3 $\sigma$ ” rule has been widely used in statistics 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 (Liu et al., 2008a). Accordingly, the maximum, mean and minimum durations of activity an  $a_i$  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 runtime is denoted as  $R(a_i)$ . As explained in Liu et al. (2008b, in press), the overall workflow completion time can be effectively estimated with the joint normal distribution of individual activity durations. Here, the probability based runtime temporal consistency model is introduced.

**Probability based runtime temporal consistency model.** At the runtime execution stage, at activity  $a_p$  where  $p \leq 1$ , the upper bound

constraint  $U(a_k, a_l)$  with the value of  $u(a_k, a_l)$ , where  $k \leq p$ , 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  $a_l$ . Temporal constraints are typically of three types, viz., upper bound, lower bound and fixed-time. An upper bound constraint between two activities is a relative time value so that the duration between them must be less than or equal to it. As discussed in Chen and Yang (in press), conceptually, a lower bound constraint is symmetrical to an upper bound constraint and a fixed-time constraint can be viewed as a special case of upper bound constraint, hence they can be treated similarly. Therefore, in this paper, we focus on upper bound constraints only.  $R(a_k, a_p)$  denotes the sum of runtime durations,  $\lambda(-3 \leq \lambda \leq 3)$  is defined as the  $\theta\%$  confidence percentile with the cumulative normal distribution function of

$$F(\mu_i + \lambda\sigma_i) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\mu_i + \lambda\sigma_i} e^{-(x-\mu_i)^2/2\sigma_i^2} \cdot dx = \alpha\%$$

with  $0 < \alpha < 100$  (Law and Kelton, 2007). Note that in this model, only one execution path is considered. However, the model could be applied to multiple paths with either repetitions (Chen and Yang, in press) or structure weights (Liu et al., in press). For example, with repetitions, for a scientific workflow containing many parallel, choice and/or mixed structures, firstly, we treat each structure as a compound activity, then, the whole scientific workflow will be an overall execution path and we can apply the results achieved in this paper to it. Secondly, for every structure, for each of its branches, we continue to apply the results achieved in this paper. Thirdly, we carry out this recursive process until we complete all branches of all structures.

##### 3.1.2. The probability range of 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 (Stroud, 2007). As depicted in Fig. 2, based on this model, the continuous probability range of (0.13%, 99.87%) represented by the shadowed area 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 Liu et al. (2008b), the overall completion time of the entire workflow instance can

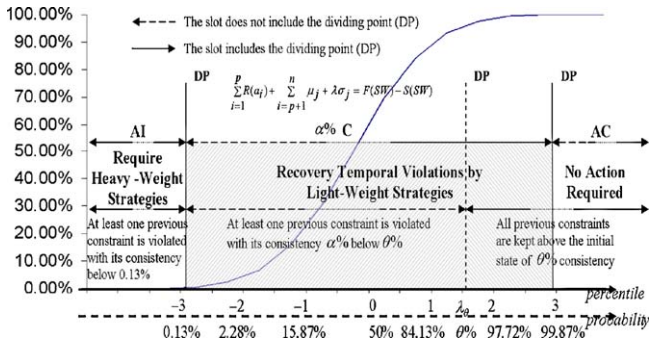


Fig. 2. The probability-based runtime temporal consistency model.

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 100–99.87%, 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 100–0.13%, i.e. 99.87%, the global temporal constraints will be violated. Apart from these two types of extreme situations, 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. Note that as can be seen in Fig. 2, the dividing point of  $\lambda_\theta$  which corresponds to the probability consistency state of  $\theta\%$  is the minimum acceptable initial temporal consistency state and it is usually specified through the negotiation between clients and service providers for setting local and global temporal constraints (Liu et al., 2008b, in press). In practice, the lower bound of  $\theta\%$  is normally set as 50%, i.e.  $\mu$ , which means temporal constraint is equal to the sum of the mean activity durations. However, ideally,  $\theta\%$  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  $\theta\%$  (i.e. for the range of ( $\theta\%$ , 99.87%)) the QoS contract still holds and thus requires no actions. But when  $\alpha\%$  is smaller than  $\theta\%$  (i.e. for the range of (0.13%,  $\theta\%$ )), light-weight exception handling strategies are required.

To conclude, with the probability based temporal consistency model, the recoverable range for temporal violations is identified as (0.13%, 99.87%). This is the range where a general exception handling framework would focus on. As for those severe temporal violations, heavy-weight exception handling strategies such as resource recruitment and workflow restructure are required which will be further investigated in the future.

### 3.2. The framework

A general exception handling framework consists of both the definition of fine-grained temporal violations and the design of handing strategies which are highly dependent to each other. First, the definition of different levels of fine-grained temporal violations

should consider the capabilities (i.e. the handling capability on temporal violations as will be defined in Definition 4) of the underlying handling strategies employed in the system. Otherwise, if a specific level of temporal violation is beyond the capability of a specific handling strategy, temporal violations cannot be handled automatically and thus deteriorates the system performance. On the contrary, if the capability of a specific handling strategy is highly above its targeting temporal violations, large exception handling cost is normally wasted. Second, in order to maximise the cost-effectiveness of handling strategies, different levels of temporal violations need to be defined in the first place so that the handling strategies with lower cost but sufficient capability can be implemented when they are detected. Therefore, to handle temporal violations in an automatic and cost-effective fashion, a framework which includes several levels of predefined fine-grained temporal violations and their corresponding handling strategies is required. To further illustrate the idea, we formally define a general exception handling framework. But first, the probability time deficit is defined.

**Definition 1 (Probability time deficit).** At activity  $a_p$ , let  $U(SW)$  be of  $\beta\%$  consistency with the probability percentile of  $\lambda_\beta$  which is below the threshold of  $\theta\%$  with the probability percentile of  $\lambda_\theta$ . Then the probability time deficit of  $U(SW)$  at  $a_p$  is defined as  $PTD(U(SW), a_p) = [R(a_1, a_p) + \theta(a_{p+1}, a_n)] - U(SW)$ . Here,  $\theta(a_{p+1}, a_n) = \sum_{k=p+1}^n (\mu_k + \lambda_\theta \sigma_k)$ .

**Definition 2 (Maximum probability time deficit).** Let  $U_1, U_2, \dots, U_N$  be the  $N$  upper bound temporal constraints and all of them cover  $a_p$ . Then, at  $a_p$ , the maximum probability time deficit is defined as the maximum of all probability time deficits of  $U_1, U_2, \dots, U_N$  is represented as  $MPTD(a_p) = \text{Min}\{PTD(U_s, a_p) | s = 1, 2, \dots, N\}$ .

The probability time deficit is defined to measure the occurring time deficit at the current checkpoint given the upper bound constraint which is set on the last activity of a scientific workflow or workflow sub-processes. The definition of the maximum probability time deficit is to consider the situation where a checkpoint is covered by multiple constraints. Clearly, only when the maximum probability time deficit is compensated can the current temporal violations be handled.

**Definition 3 (A general exception handling framework).** Given the requirements of automation and cost-effectiveness, a general exception handling framework can be denoted as  $F = \{(V_i, HS_i) | i = 1, 2, 3, \dots, K\}$ .

Here,  $\{V_i | i = 1, 2, 3, \dots, K\}$  denotes  $K$  different levels of temporal violations where  $V_i < V_{i+1}$  means that the maximum probability time deficits occurring with level  $V_i$  is smaller than that with level  $V_{i+1}$  (denoted as  $MPTD(V_i) < MPTD(V_{i+1})$ ), namely level  $V_{i+1}$  is more severe than level  $V_i$ ;  $\{HS_i | i = 1, 2, 3, \dots, K\}$  denotes  $K$  automatic handling strategies of different capabilities where  $HS_i < HS_{i+1}$  means that the capability of  $HS_i$  in compensating the time deficits is weaker than that of  $HS_{i+1}$  (denoted as  $CAP(HS_i) < CAP(HS_{i+1})$ ), namely  $HS_i$  is a less capable handling strategy than  $HS_{i+1}$ . Furthermore, for each pair of  $(V_i, HS_i)$ , it means that  $HS_i$  is capable of compensating the occurring time deficits  $TD(V_i)$  for level  $V_i$  temporal violations. Here, without losing generality, we can assume that the handling strategy with higher capability normally has heavier weight (e.g. computation cost and time overhead). Therefore, in such a general exception handling framework, each level of temporal violation is addressed by the handling strategy of sufficient capability yet with the least cost among the available set of handling strategies.

The capability of handling strategies is defined as follows.

**Definition 4 (Handling capability on temporal violations).** For a specific handling strategy  $HS_i$ , its handling capability on temporal

violations is defined by a capability lower bound of  $\lambda_i$ .  $\lambda_i$  stands for a normal confidence percentile defined in the probability based runtime temporal consistency model and corresponds to a probability value of  $\theta_i\%$ . The probability lower bound of  $\lambda_i$  defines that strategy  $HS_i$  can only handle the temporal violations with probability consistency states which are above  $\theta_i\%$ .

For example, as shown in Fig. 1, the probability range of (0.13%, 99.87%) can be divided into a total of  $K$  consecutive intervals by  $K$  dividing points of  $\{\lambda_1, \lambda_2, \lambda_3, \dots, \lambda_K\}$  which stand for the capability lower bound of each handling strategy. The value of each capability lower bound can be specified through the empirical function provided as follows.

**Definition 5** (Empirical function for capability lower bound). For a specific exception handling strategy  $HS_i$ , its capability lower bound  $\lambda_i$  can be estimated by its ability in minimising the makespan of workflow segments. Assume we have a workflow segment  $ws$  and its duration follows  $N(\mu, \sigma)$ . After the implementation of  $HS_i$ , the mean duration can be reduced to  $\mu'$ . Hence, the effectiveness of strategy  $HS_i$  on minimising the makespan can be measured by the value of *CompensationRatio* (the time deficit compensation ratio) which can be calculated as follows:

$$\text{CompensationRatio} = 100\% - \frac{\mu'}{\mu} \quad (1)$$

Here, the capability lower bound  $\lambda_i$  for strategy  $S_i$  is defined as follows:

$$\alpha_i\% = \theta\% - \text{CompensationRatio} \quad (2)$$

where  $\lambda_i$  is the  $\alpha_i\%$  normal percentile and  $\theta\%$  is the minimum acceptable initial temporal consistency.

The purpose of the empirical function is to estimate the capability lower bound for each handling strategy based on the amount of reduced activity durations within the recoverable probability range. Its rationale is explained by the proof as follows.

**Proof.** Here we assume for  $m$  activities  $\{a_1, a_2, \dots, a_m\}$ , their duration models are  $\{N(\mu_i, \sigma_i) | i = 1, \dots, m\}$ . Hence, their total completion time, namely the makespan of the segment, can be estimated by the weighted joint normal distribution  $N(\mu, \sigma)$  (Liu et al., 2008b). Here, we define the relationship between  $\mu$  and  $\sigma$  as  $\sigma = \rho \cdot \mu$ . Therefore, according to the  $3\sigma$  rule, the makespan range  $(\mu - 3\sigma, \mu + 3\sigma)$  is turning into  $((1 - 3\rho)\mu, (1 + 3\rho)\mu)$  with the probability range of (0.13%, 99.87%). According to Definition 5, given a specific exception handling strategy  $S_i$ , the amount of reduced makespan  $M(ws) - m(ws)$  equals to *CompensationRatio*  $\times M(ws)$ , i.e. *CompensationRatio*  $\times \mu$ . Meanwhile, since the initial probability consistency state  $\theta\%$  is normally above 50%, i.e.  $\mu$  and the maximum activity durations are defined as  $\mu + 3\sigma$ , the range for the occurring probability time deficits are defined as  $(0, 3\sigma)$ , i.e.  $(0, 3\rho\mu)$ . Therefore, the ratio between the reduced makespan and the maximum probability time deficit is  $\gamma = \text{CompensationRatio} \times \mu / 3\rho\mu = \text{CompensationRatio} / 3\rho$ . Here, considering the highly dynamic performance of the underlying infrastructures in scientific workflow environments,  $\rho$  is often a relatively large value within the valid range of  $(0, 1/3)$ . Since makespan are positive values in real world, the upper bound for  $\rho$  is  $1/3$  (Law and Kelton, 2007). Otherwise, the lower limits of the makespan  $(\mu - 3\rho)$  will become negative values. Therefore, to consider the worst case, i.e.  $\rho = 1/3$ , *CompensationRatio* /  $3\rho$  becomes *CompensationRatio*, i.e.  $\gamma$  is equal to the *CompensationRatio*.

Here, we assume the current temporal consistency is  $\alpha\%$  (with a minimum value of 0.13%) and the minimum temporal consistency is  $\theta\%$  (with a maximum value of 99.87%). Since  $\gamma$  is defined based on the maximum probability time deficits, if  $\gamma$  is equal to or larger than  $\theta\% - \alpha\%$ , then the current temporal violation can be recovered since the reduced activity durations will be large enough to adjust

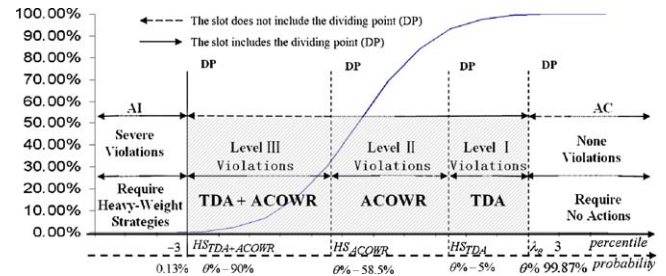


Fig. 3. Fine-grained temporal violations based on three exception handling strategies.

$\alpha\%$  back to or above  $\theta\%$ . Otherwise, if  $\gamma$  is smaller than  $\theta\% - \alpha\%$ , then the current temporal violation cannot be recovered but the temporal consistency state will be adjusted back to  $\alpha\% + \gamma$ . For the worst case,  $\alpha\%$  is equal to the minimum value, i.e. 0.13%. Then, given an exception handling strategy  $HS_i$  with  $\gamma$ , the temporal consistency state can be adjusted back to  $0.13\% + \gamma$ . In other words, the current temporal violation can be recovered as long as its temporal consistency is equal to or above  $\theta\% - (0.13\% + \gamma)$ . For the ease of application, 0.13% is rounded off to 0. Therefore, according to Definition 4, the capability lower bound for  $HS_i$  with  $\gamma$  is defined as  $\theta\% - \gamma$ . As mentioned above,  $\gamma$  is equal to the *CompensationRatio*, i.e.  $\theta\% - \gamma = \theta\% - \text{CompensationRatio}$ , and hence Definition 5 holds. □

In practice, the value of  $\rho$  could be adjusted according to the actual system performance which is normally smaller than  $1/3$ , i.e. the upper bound value. However, to propose a general exception handling framework, the worst case is considered in our definitions. Therefore, if the actual value of  $\rho$  is smaller than  $1/3$ , the actual capability lower bound will be higher than the one defined in our strategy. In such a case, the cost-effectiveness of the framework will be affected since the handling strategy with higher cost (with higher capability) may be selected to handle the temporal violations which can be actually handled by the one with lower cost. However, since without any priori knowledge, it is difficult, if not impossible, to select a reasonable  $\rho$  at the initial stage. One of the practical ways is to set  $\rho$  as the upper bound value (i.e.  $1/3$ ) at the beginning and then dynamically adjust the value periodically according to the real system performance.

Here, we present an example handling framework. Given a set of three handling strategies  $HS\{HS_1, HS_2, HS_3\}$  with the *CompensationRatio* of (30%, 50%, 80%), and the initial temporal consistency state is 90%, i.e.  $\gamma$  is equal to 90%. The three dividing points are hence defined as 90% minus the *CompensationRatio* of each strategy respectively, i.e. (60%, 40%, 10%) according to Definitions 4 and 5. Therefore, according to Definition 3, three levels of fine-grained temporal violations are defined as level I ( $90\% > \alpha\% \geq 60\%$ ), level II ( $60\% > \alpha\% \geq 40\%$ ) and level III ( $40\% > \alpha\% \geq 10\%$ ). In such a case, since  $HS_3$  cannot handle the violations with probability consistency states below 10%, the rest of the probability range, i.e. (0.13%, 10%) is hence merged to AI (Absolute Inconsistency). In such an example, a handling framework which consists of three levels of fine-grained temporal violations and three corresponding handling strategies is hence built up.

Based on such a general exception handling framework, the cost on handling temporal violations in scientific workflows can be minimised while satisfactory temporal QoS is guaranteed. However, since the capability and weight of handling strategies are normally not in a linear relationship, it is difficult to build up such a framework in the real world. The purpose of presenting the definition of a general framework here is to investigate the relationship between the definition of fine-grained temporal violations and the design of

handling strategies. Meanwhile, it can serve as a guideline for our investigation on solving these two basic problems.

#### 4. An example implementation of the framework

Based on the general exception handling framework defined in Section 3, this section presents an automatic and cost-effective exception handling framework which serves as a representative example.

##### 4.1. Example framework overview

In our example framework, the exception handling strategies include TDA (Time Deficit Allocation) (Chen and Yang, 2007), ACOWR (Ant Colony Optimisation based two-stage Workflow local Rescheduling) (Liu et al., 2010), and the combined strategy of TDA+ACOWR. As will be presented in Section 5.3.2 for the experimental results, the values of *CompensationRatio* for them are 5%, 58.5% and 90% respectively. Therefore, according to Definition 5, their capability lower bounds are  $\theta\% - 5\%$ ,  $\theta\% - 58.5\%$  and  $\theta\% - 90\%$  correspondingly where  $\theta\%$  is the initial temporal consistency. As discussed in Section 3,  $\theta\%$  is normally set around 84.13%, i.e.  $\mu + \sigma$  where  $\mu$  and  $\sigma$  are the mean and the standard deviation of the overall workflow execution time respectively, to represent a reasonable QoS requirement (Liu et al., 2008b, in press). In this paper, to investigate the performance of our strategy in a more general yet demanding scenario, the upper bound for  $\theta\%$  is set as 90%, i.e.  $\lambda_{\theta}$  is equal to 1.28. In other words, the probability of meeting the temporal constraints along scientific workflow execution are required to be maintained no smaller than 90% at all time. For those scenarios where  $\theta\%$  is higher or lower than 90%, the capability lower bounds for the exception handling strategies will be adjusted accordingly and so are the definitions for different levels of temporal violations.

According to the above results, and Definitions 3–5, three levels of fine-grained temporal violations including level I, level III and level III are defined. As can be seen in Fig. 3, 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 local and global temporal constraints (Liu et al., 2008b), i.e. temporal QoS contracts. In practice,  $\theta\%$  is normally around or above 84.13%, i.e.  $\mu + \sigma$ , which denotes reasonable confidence for on-time completion. But in this paper, we set the upper bound of  $\theta\%$  as 90% in order to investigate a more general scenario. If current temporal consistency state of  $\alpha\%$  is larger than  $\theta\%$ , the QoS contract still holds and thus there is no temporal violation. Besides the dividing point of  $\theta\%$ , the other three dividing points are  $\theta\% - 5\%$  (denoted as  $HS_{TDA}$ ),  $\theta\% - 58.5\%$  (denoted as  $HS_{ACOWR}$ ) and  $\theta\% - 90\%$  (denoted as  $HS_{TDA+ACOWR}$ ), which are the capability lower bound for the three exception handling strategies respectively. Therefore, according to Definition 3, three levels of temporal violations are defined. Specifically, as shown in Fig. 3, level I temporal violations are those temporal consistency states in the range of  $(\theta\%, \theta\% - 5\%]$ , level III temporal violations are those temporal consistency states in the range of  $(\theta\% - 5\%, \theta\% - 58.5\%]$ , level III temporal violations are those temporal consistency states in the range of  $(\theta\% - 58.5\%, \theta\% - 90\%]$ .

To conclude, three levels of fine-grained temporal violations are defined within the recoverable probability range. Note that since our definition of fine-grained temporal violations is fully compatible with that of multiple-state based temporal consistency model, the state-of-the-art checkpoint selection strategies (Chen and Yang, in press) which deal with the detection of temporal violations can be applied directly. Therefore, in this paper, we only focus on

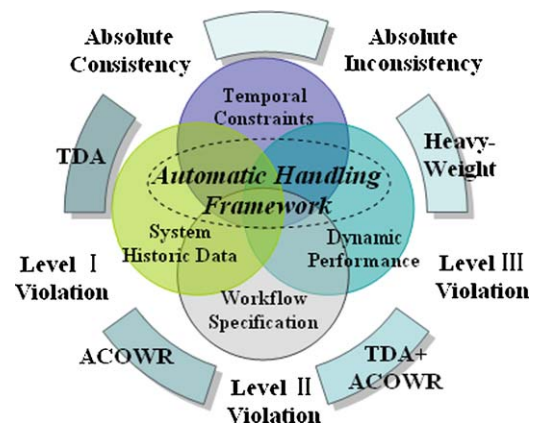


Fig. 4. Overview of the example exception handling framework.

how to handle these temporal violations rather than how to detect them.

Furthermore, based on the three fine-grained levels of temporal violations defined above, the overview of our example exception handling framework is presented in Fig. 4. The inner four circles stand for the basic factors concerned with the handling strategies for temporal violations which include the local and global temporal constraints, the dynamic performance of underlying resources, the specifications for scientific workflows, and the system historic data for time attributes. The outer layer describes five different types of temporal consistency states in a clockwise order according to the descending severity of temporal violations: AI, level III, level II, level I and AC. The links between different temporal consistency states represent the corresponding handling strategies in order to handle the current temporal violation. As shown in Fig. 4, for AC, no action is required since there are no temporal violations. For AI, heavy-weight exception handling strategies are required. In this paper, we focus on the handling strategies for the three levels of temporal violations defined in the recoverable probability range. Specifically, TDA is facilitated for handling level I temporal violations; ACOWR is facilitated for handling level II temporal violations; the combined strategy of TDA and ACOWR (denoted as TDA+ACOWR) is facilitated for handling level III temporal violations.

As discussed in Chen and Yang (2007), the basic idea of TDA is to automatically utilise the expected time redundancy of the subsequent workflow segments to compensate the current time deficits. ACOWR, as one type of workflow rescheduling strategies, is to tackle the violations of QoS constraints through optimising the current plan for task-to-resource assignment (Liu et al., 2010). When temporal violations are detected, these strategies can be realised automatically without human interactions. Furthermore, TDA produces only a small amount of calculations. ACOWR, as a type of metaheuristic searching algorithm, consumes more yet acceptable computing time but without recruiting additional resources. Therefore, as will also be verified through the simulation experiments demonstrated in Section 5, TDA, ACOWR and the combined strategy of TDA and ACOWR are effective candidates for handling temporal violations given the requirements of both *Automation* and *Cost-effectiveness*. The detailed algorithms are presented next.

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

In this section, we will present the algorithms for the three 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

**Strategy I : Time Deficit Allocation**

**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  $M(\mu_i, \sigma_i)$ ;  
**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  $U'(a_i) | i = p+1, \dots, p+m$

---

Fig. 5. Algorithm for TDA.

subsequent workflow activities (Chen and Yang, 2007, 2010; Liu et al., in press). In our example framework, TDA is responsible for handling level I temporal violations. The pseudo-code for TDA is presented in Fig. 5.

The actual process of TDA is to borrow the expected time redundancy of the next workflow segment to compensate the current time deficit and then allocate the time deficit to the subsequent activities. The first task of TDA is to calculate the expected time redundancy of the next workflow segment (Line 1 of Fig. 5). 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 of Fig. 5). 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 Chen and Yang (2007) (Lines 2–4 of Fig. 5). After that, re-assigned temporal constraints for each activity are returned (Line 5 of Fig. 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

In ACOWR, “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.

To handle temporal violations, the key optimisation objective for ACOWR is to maximise the compensation time. However, if we only focus on the speed up of the workflow instance where temporal violations are detected, the completion time of other activities such as the segments of other workflow instances and

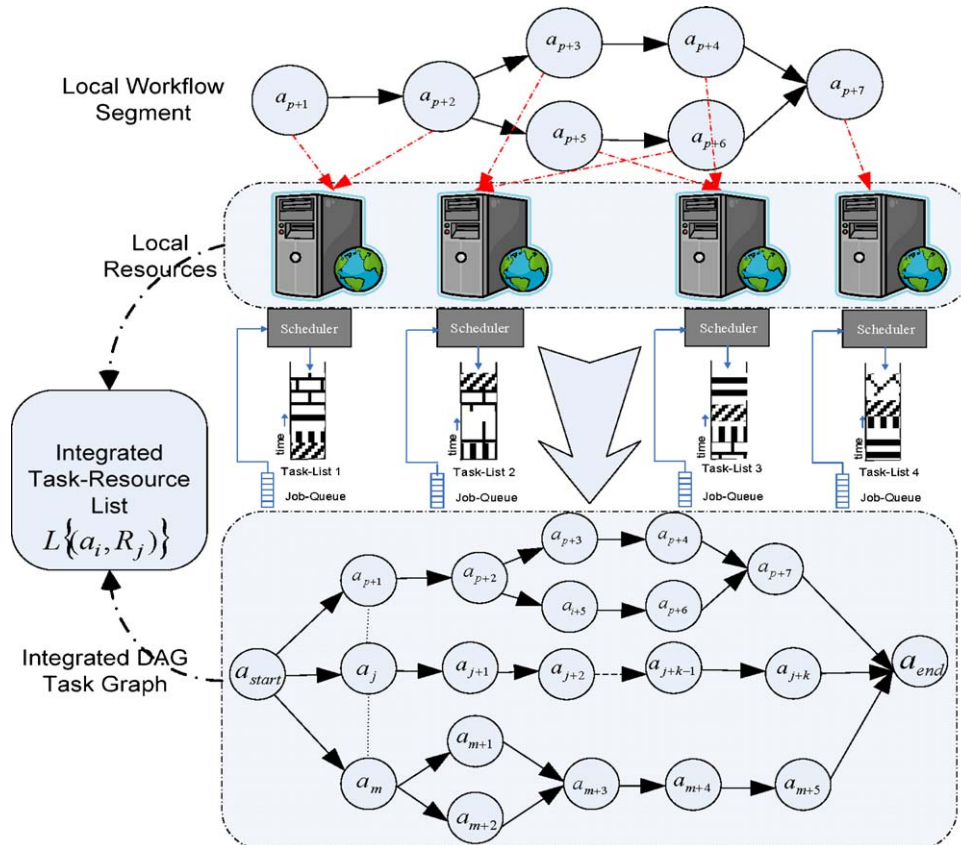


Fig. 6. An example of integrated task resource list.

ordinary non-workflow tasks, could be delayed and may violate temporal constraints of their own, if any. Therefore, a balance between time deficit compensation and the completion time of other activities needs to be considered. Otherwise, the overall efficiency of scientific workflows will be significantly deteriorated. As for local rescheduling, it is designed for the requirement of *Cost-effectiveness*. ACOWR only utilises existing resources which are currently deployed in the system instead of recruiting additional resources. Meanwhile, unlike global rescheduling which modifies the global task-resource list for the entire workflow instance, ACOWR only focus on the local workflow segment and optimise the integrated task-resource list. Here, similar as in TDA, the local workflow segment is the workflow segment that between the next activity of the checkpoint and the end activity of the next local temporal constraint. In our strategy, as depicted in Fig. 6, the integrated task-resource list is an integrated collection of local resources and the integrated DAG task graph which defines the precedence relationships of all the activities in the local workflow segment and their co-allocated activities. Here, co-allocated activities are those which have been allocated to the same resources. For example, the local workflow segment contains activity  $a_{p+1}$  to  $a_{p+7}$  and they are allocated to four different resources  $R_1$  to  $R_4$ . Each resource maintains a local task-list by its scheduler given its input job-queue. When ACOWR starts, the workflow management system will acquire the current task-list of  $R_1$  to  $R_4$  and can automatically combine them into an integrated DAG task graph which consists of all the tasks, say a total of  $n$  tasks, by assigning a pseudo start activity  $a_{start}$  and pseudo end activity  $a_{end}$ . Hence, an integrated task-resource list  $L\{(a_i, R_j) | i = p + 1, \dots, p + n, j = 1, 2, 3, 4\}$  is built and ready to be optimised by ACOWR.

The pseudo-code for ACOWR is presented in Fig. 7. Since the algorithm for the ACO based searching stage is similar to the one described in Chen et al. (2007), here we focus on describing the process but omitting the detailed discussion for algorithms and parameters. The algorithm has five input parameters: the time deficit detected at the checkpoint; the integrated task-resource list; the DAG task graphs which define the precedence relationships between tasks; the normal distribution models for activity durations; and resources with their execution speed and the cost per time unit. As shown in Fig. 7, the first searching stage is to optimise the overall execution time and cost for the integrated task-resources list through ACO (Lines 1–10). The ACO algorithm starts from initialisation of pheromone and all parameters (Line 1). In Chen et al. (2007), two types of pheromone, i.e.  $d\tau_{ij}$  and  $c\tau_{ij}$ , are defined. Here,  $d\tau_{ij}$  denotes the desirability of mapping task  $a_i$  to resource  $R_j$  from the perspective of execution time while  $c\tau_{ij}$  denotes the desirability from the perspective of execution cost. Afterwards, the ACO based searching process iterates until the stopping condition, e.g. the maximum iteration times, is satisfied. During each iteration, a group of ants needs to be initialised first (Lines 3–4). Each ant starts with selecting one of the heuristics from duration-greedy, cost-greedy or overall-greedy (Line 3). Then, the tackling sequence which arranges the order of tasks is built based on the input DAG task graph (Line 4). During the solution construction process (Lines 5–8), each activity is allocated to a specific resource according to its bias  $B_{ij}$  which is based on the value of pheromones and the heuristic information (Line 6). Meanwhile, after a specific choice of resources, the earliest start time  $est$  of the current activity is compared with the earliest end time  $eet$  of its predecessors to determine whether the current schedule can satisfy the precedence relationships defined in the DAG task graph. After a successful resource allocation, the  $est$  and  $eet$  for its subsequent activities are updated (Line 7). Here, a local updating process is conducted to decrease the local pheromone of  $d\tau_{ij}$  and  $c\tau_{ij}$  so that the following ant can have a higher proba-

### Strategy II: ACOWR

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

**Output:** Rescheduled task-resource list

---

```

// Stage 1: Optimising the overall execution time and cost for the
integrated task-resource list through ACO algorithm
//Initialisation of pheromone and other parameters for ACO algorithm
1) INITIALISATION(ACO);
2) While (stopping condition is not met)
{
// initialise each ant
for each ant
{
// select heuristics from duration-greedy, cost-greedy and overall-greedy
3) SELECTION(Heuristics);
// build tackling sequence TS based on the input DAG task graphs
4) BUILDING(TS, DAGs);
}
// construct solutions
5) While (not end of TS)
{
// choose the resource for the next activity based on its earliest start time (est),
earliest end time (eet), and the bias of  $B_{ij}$  (mapping resource  $R_j$  to activity  $a_i$ )
6) CHOOSE( $a_i, a_i, est, a_i, eet, R\{R_j\}, B_{ij}$ );
// update the est and eet for all the subsequent activities;
7) UPDATE(EST, EET);
// update local pheromone for both duration and cost
8) LOCALUPDATE( $d\tau_{ij}, c\tau_{ij}$ );
}
// update the global pheromone based on the makespan and cost of the best-so-
far solution
9) GLOBALUPDATE( $d\tau_{ij}, c\tau_{ij}, makespan^{bs}, cost^{bs}$ );
// return the best-so-far solution and record into the SolutionSet
10) Return(Solution, SolutionSet)
}
// Stage 2: Searching for the BestSolution from SolutionSet
11) 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
12) COMPARE(Solution.ct, TD( $a_p$ ));
// compare all remaining Solution and set BestSolution as the one with the
minimum cost
13) BestSolution=MIN(Solution.cost);
}
14) Return the BestSolution;
// return the rescheduled integrated task-resource list and deploy
15) DEPLOY(L)

```

---

Fig. 7. Algorithm for ACOWR.

bility of choosing other resources (Line 8). Evidently, the purpose of local updating is to enhance the diversity of the ACO algorithm. By contrast, after all ants have built their individual solutions, a global updating process is conducted to increase the pheromone along the path for the best-so-far solution so that the subsequent ants have higher probability to choose the same scheduling plan (Line 9). Therefore, the purpose of global updating is to reinforce the best-so-far solution in order to speed up the convergence of the ACO algorithm. Finally, at the end of each iteration, the best-so-far solution is returned and added into the *SolutionSet* which serves as the input for the second searching stage (Line 10).

In the second searching stage, the *BestSolution* which can compensate the time deficit detected at the checkpoint is retrieved from the *SolutionSet* (Lines 11–14). The compensation time of each solution is first compared with the time deficit to filter out unsuccessful solutions (Line 12). Then, the *BestSolution* is defined as the best solution with the minimum cost among all the successful solutions (Line 13). Finally, the *BestSolution*, i.e. the solution which can handle the temporal violations while having the minimal cost and the minimal affect on the completion time of other tasks, is returned as the rescheduled integrated task-resource list (Line 14) and ready to be deployed (Line 15).

**Strategy III: TDA+ACOWR**


---

**Input:** Time deficit detected at activity  $a_p$   $TD(a_p)$ ;  
 The next workflow segment  $WS(a_{p+1}, a_{p+2}, \dots, a_{p+m})$ ;  
 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)  $TR(a_{p+1}, a_{p+m}) = U(a_{p+1}, a_{p+m}) - \sum_{i=p+1}^{p+m} (\mu_i + \lambda_{\theta} * \sigma_i)$ ;  
 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)  $TD(a_p) = TD(a_p) - BestSolution.ct$ ;  
 6) if  $TD(a_p) > 0$   
**// read in the second next workflow segment**  
 {  
 7)  $WS = WS(a_{p+m+1}, a_{p+m+2}, \dots, a_{p+m+m'})$ ;  
**// update all the required information**  
 8)  $UPDATE(L\{\}, DAG\{\}, M\{\}, R\{\})$ ;  
 }  
 }  
**// return the rescheduled integrated task-resource list and deploy**  
 9) DEPLOY(L)

---

Fig. 8. Algorithm for TDA + ACOWR.

**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 stronger exception handling capability. The pseudo-code for ACOWR is presented in Fig. 8.

The combined strategy starts with the first stage of TDA (Lines 1–2). 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 (Lines 3–8). Here, ACOWR is called for the first time to compensate the time deficit with the best solution for the next workflow segment (Lines 4–5). 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 (Lines 7–8). 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 5, 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).

**4.3. Discussion**

Besides the three exception handling strategies introduced in this section, there are many others available in practice. Take the rescheduling strategy as an example, there are many other meta-

heuristics based algorithms such as GA (genetic algorithm), PSO (particle swarm optimisation), SA (simulated annealing). Moreover, there are many other simple, manual or semi-automatic, but heavy weight strategies such as resource reservation, resource recruitment and workflow restructure (Buhr and Mok, 2000; Cooper et al., 2004; Hagen and Alonso, 2000; Prodan and Fahringer, 2008; Russell et al., 2006a). Therefore, in this paper, we cannot and do not intend to investigate all the exception handling strategies. Our focus is on how to build up an automatic and cost-effective exception handling framework based on existing exception handling strategies. Clearly, different specific exception handling frameworks can be built up given Definition 3 on a general exception handling framework. Our example framework presented in this section is mainly for the case study purpose. In the future, more exception handling strategies could be investigated in order to build up more specific exception handling frameworks to meet the requirements of different application scenarios.

**5. Evaluations on example framework**

In this section, we evaluate the performance of the example framework to demonstrate the effectiveness of our general exception handling framework. In a qualitative fashion, we can claim that our example framework satisfies the two basic requirements of *Automation* and *Cost-effectiveness*.

**Automation:** Based on our previous work on checkpoint selection and temporal verification (Chen and Yang, in press), different levels of temporal violations can be automatically detected in an efficient fashion. Afterwards, given the three exception handling strategies in our example framework, TDA, ACOWR and TDA + ACOWR, as presented in Section 4.2, can all be implemented automatically to tackle different levels of temporal violations without the needs of human intervention. Therefore, our example framework satisfies the basic requirement of *Automation*.

**Cost-effectiveness:** As discussed in Section 3.2 for the general exception handling framework, a framework is cost-effective if a heavier weight exception handling strategy has a higher handling capability than a lighter weight one. In such a case, a specific level of temporal violation can be handled by the strategy with the least cost among all the strategies with sufficient capability. As presented in Section 4.2, it is evident that the cost of TDA, ACOWR and TDA + ACOWR is on an increasing trend and so are their handling capabilities. In such as case, TDA, ACOWR and TDA + ACOWR can each responsible for their corresponding levels of temporal violations as defined in our example framework. Therefore, our example framework satisfies the basic requirement of *Cost-effectiveness*.

In the following subsections, we will demonstrate the comprehensive experimental results to evaluate the effectiveness of our example framework in a quantitative fashion. First, we introduce the SwinDeW-G simulation environment. Second, we present the detailed experiments settings. Afterwards, the *CompensationRatio* of ACOWR is evaluated in particular which denotes its handling capability. Finally, the effectiveness of our example exception handling framework is evaluated through large scale simulation experiments where the violation rates of global temporal constraints and local temporal constraints are observed and compared with other strategies.

**5.1. Simulation environment**

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 (Yang et al., 2007). An overall picture of SwinGrid is depicted in Fig. 9 (bottom plane).

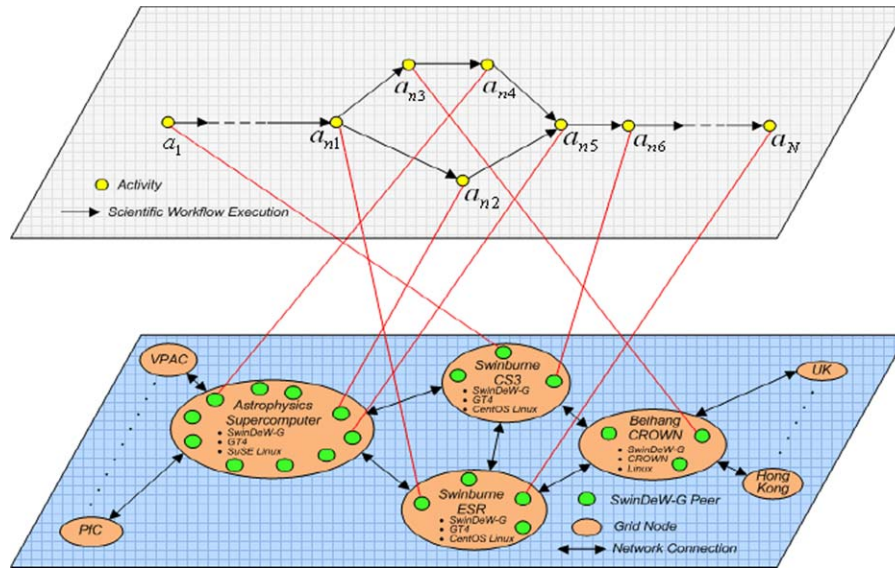


Fig. 9. Overview of SwinDeW-G environment.

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. Besides, the CROWN Node is also connected to some other nodes such as those in Hong Kong University of Science and Technology, and University of Leeds in UK. The Swinburne Astrophysics Supercomputer Node is cooperating with PFC (Australian Platforms for Collaboration) and VPAC (Victorian Partnership for Advanced Computing). Currently, SwinDeW-G is deployed at all

primary hosting nodes as exemplified in the top plane of Fig. 9. For example, the Swinburne Astrophysics Supercomputer Node (<http://astronomy.swin.edu.au/supercomputing/>) comprises 145 Dell Power Edge 1950 nodes each with: 2 quad-core Clovertown processors at 2.33 GHz (each processor is 64-bit low-volt Intel Xeon 5138), 16 GB RAM and 2 × 500 GB drives. In SwinDeW-G, a scientific workflow is executed by different peers that may be distributed at different grid nodes. As shown in Fig. 9, each grid node can have a number of peers, and each peer can be simply viewed as a grid service. In the top plane of Fig. 9, we show a sample of how a grid workflow can be executed in the simulation environment.

To improve the overall workflow QoS, temporal verification is an important function being implemented in SwinDeW-G. SwinDeW-G currently supports constraint setting at build-time, dynamic checkpoint selection and temporal verification at runtime (Chen and Yang, in press; Liu et al., 2008b). After running SwinDeW-G for a period of time, statistical analysis can be applied to accumu-

Table 1  
Experimental settings for workflow rescheduling.

Round	A	W	R
Setting for input parameters			
Workflow process setting	Randomly generated DAG task graphs with a random size of (3–5) activities for each local workflow segment		
Duration distribution setting	Duration distribution of $a_j$ is $N(\mu_j, \sigma_j^2)$ where $\mu_j = \text{random}(30, 3000)$ and $\sigma_j = 33.3\% \times \mu_j$		
Resource setting	$R(R_i, ES(R_i), COST(R_i))$ , resource $R_i$ with the execution speed of $ES(R_i) = \text{random}(1, 5)$ where the mean duration of $a_j$ on resource $R_i$ is $\mu_j/ES(R_i)$ and the resource price is $COST(R_i) = ES(R_i) + \text{random}(1, 3)$		
Time deficits setting	The time deficits are randomly set as 50–90% of the mean durations of the local workflow segments		
Integrated task-resource list setting	$L(a_i, R_j)$ is defined with $A$ (the number of activities), $W$ (the number of workflow instances) and $R$ (the number of local resources)		
Setting for A, W and R			
1	50	6	3
2	80	10	
3	120	15	6
4	150	20	8
5	180	25	10
6	200	28	12
1	220	30	15
8	250	35	16
9	280	40	18
19	300	50	20

lated system logs to obtain probability attributes such as activity duration distribution models (Liu et al., 2008a). The automatic handling framework is being integrated as an important component of the scientific workflow monitoring and management tool in SwinDeW-G which supports automatic constraint setting, checkpoint selection, temporal verification and the handling of temporal violations.

## 5.2. Experimental settings

Based on the statistics obtained from the historic data for the pulsar searching workflow presented in Section 2.1 and also a weather forecast example as presented in Liu et al. (in press), representative settings are adopted in our simulation experiments. The weather forecast workflow is also a typical data and computation intensive scientific workflow which deals with the processing of large-size meteorology data collected from many geographically distributed equipments such as radars and satellites. Specifically, the process definitions are generated according to the example workflows; as for these time attributes, e.g. activity durations and time deficits, their average values are specified with the statistics but deliberately with larger ranges and/or variances in order to simulate more general system environments.

The handling capability of our framework mainly depends on the performance of ACOWR. Therefore, simulation experiments are first conducted to evaluate ACOWR and obtain its *CompensationRatio* as defined in Definition 5. Afterwards, with the implementation of our example framework, various simulation experiments are conducted on different sizes of scientific workflows to verify the effectiveness in the handling of both local and global temporal violations. 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 (genetic algorithm) based workflow rescheduling strategy where their results on the *CompensationRatio* are very close to each other, i.e. their handling capabilities are similar. However, due to the space limit, in this paper, we only focus on the handling capability of ACOWR while omitting other details here. More comprehensive experimental results and all the Java codes can be found online.<sup>1</sup>

### 5.2.1. Settings for workflow rescheduling

To evaluate the performance of ACOWR, the basic experiment settings for workflow rescheduling are presented in Table 1. Similar to the one shown in Fig. 6, workflow process models in the integrated task-resource list are randomly generated as DAG task graphs with a random size of 3–5 activities for each local workflow segment. The mean duration of each task is randomly selected from 30 to 3000 basic time units and its standard deviation is defined as 33% of its mean (a large standard deviation for valid normal distribution models where the samples are all positive numbers according to the “3 $\sigma$ ” rule) to represent the highly dynamic performance of underlying resources. Each resource contains three attributes including resource ID, the execution speed and the execution cost. Here, the execution speed is defined as an integer from 1 to 5 where the execution time is equal to the mean duration divided by the execution speed. In each of the ten experiment scenarios, half of the resources are with the speed of 1 and the others are with a random speed from 1 to 5. To simplify the setting, the price of each resource (for every 60 basic time units) in our experiment is defined as the execution speed plus a random number ranging from 1 to 3. For example, if a task is allocated to a resource with the execution speed of 2, then 2 basic units plus additional random

1–3 basic units, e.g. 4 basic cost units, will be charged for every 60 basic time units consumed on such a resource (namely the price of the resource is 4). The time deficits are set as a random percentage of the mean durations of the local workflow segments in the range of (50%, 90%). Based on the “3 $\sigma$ ” rule, such a setting actually poses a very high demand for the performance of the workflow rescheduling strategy. As for the integrated task-resource list, it is defined with three attributes, being  $A$  (the number of total activities),  $W$  (the number of workflow instances) and  $R$  (the number of resources). Specifically, the number of total tasks ranges from 50 to 300 including both workflow and non-workflow activities. The number of workflow segments increases accordingly from 5 to 50. The number of resources is constrained in a range of 3–20 since high performance resources in scientific workflow systems usually maintain long job queues.

### 5.2.2. Settings for workflows and temporal violations

After the evaluation of ACOWR, we further implemented our example 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 where the activity durations are generated by normal distribution models with the same settings as for workflow rescheduling.<sup>2</sup> For every group of 20–50 activities, an upper bound temporal constraint is assigned. The strategy for setting temporal constraint is adopted from the work in Liu et al. (2008b) 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 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 example framework (denoted as Framework where the three exception handling strategies as presented in Section 4 will be implemented automatically according to the levels of temporal violations). In TDA + ACOWR, the maximum iteration times for ACOWR are set as 2. 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 round of experiment is executed for 100 times to get the average violation rates.

### 5.2.3. Parameter settings for ACOWR

In ACOWR, 50 new ants are created in each iteration. Since the reduction of both total makespan and total cost of the integrated task-resource list are considered in the first searching stage of ACOWR, half of them are created as duration-greedy and another half as cost-greedy. The maximum iteration times are set as 1000 and the minimum iteration times are 100. The weights of pheromone and heuristic information are set as 1 and 2 respectively. The probability of selecting the implementation with the largest value of  $B_{ij}$  is 0.8. Local pheromone updating rate is 0.1 and the global pheromone updating rate is also 0.1.

<sup>2</sup> Different distribution models such as normal, uniform, exponential and a mixed of them have been employed in our simulation experiments, the simulation results are similar. Details can be found online.

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

Generally speaking, given longer running times, metaheuristic searching algorithms may produce better solutions (i.e. better effectiveness). However, large time overhead (i.e. low efficiency) is not acceptable in the scenario of handling temporal violations at workflow runtime. Therefore, to make a trade-off between effectiveness and efficiency, we design a compound stopping condition with four parameters: the minimum iteration times, the maximum iteration times, the minimum increase of optimisation rate on time (the increase of optimisation rate on time: the minimum makespan of last iteration subtracts the minimum makespan of the current iteration and then divided by the one of the last iteration), and the minimum increase of optimisation rate on cost (similar to that of makespan). Specifically, the evolutionary process iterates at least a minimum of 100 times. After 100 times iterations, the iteration will stop on condition that the maximum iteration times are met; or the increase of optimisation rate on time is less than 0.02; or the increase of optimisation rate on cost is less than 0.02.

Fitness value is the fundamental measurement for a candidate solution. In our strategy, we seek balanced scheduling plans where total makespan and total cost can be optimised in an unbiased fashion. Therefore, the function for the fitness value is defined as follows:

$$\text{FitnessValue}(\text{TotalMakespan}, \text{TotalCost}) = (\text{TotalMakespan} + 3 \times \text{TotalCost})^{-1} \quad (2)$$

where *TotalMakespan* and *TotalCost* are the total makespan and total cost of a candidate solution respectively.

Here, the fitness value is defined as a reciprocal relation to the sum of total makespan and total cost since the smaller the sum, the better the solution, i.e. its fitness value. Meanwhile, note that the *TotalCost* is multiplied by a factor of 3. Since the basic time unit and the basic cost unit are usually different in the real world. For example, in a scientific workflow system, if the basic time unit is one second and the basic cost unit is one dollar, the value of total makespan is normally much higher than that of total cost. For instance, for the use of Amazon EC2 standard on-demand instances service (<http://aws.amazon.com/ec2/pricing/>), the default price is \$0.12 per hour (i.e. 3600 s). Therefore, in order to adjust the makespan and the cost to the same order of magnitude, a factor is usually needed to multiple the value of cost. This factor can be selected through empirical study on the historic data or through simulation experiments. In our experiments, we have tried different candidates for the factor such as 2, 3, 5, and 8. Based on the experimental results, we find out that 3 is one of the best candidates which can adjust the makespan and the cost to the same order of magnitude for most of cases.

### 5.3. Experimental results

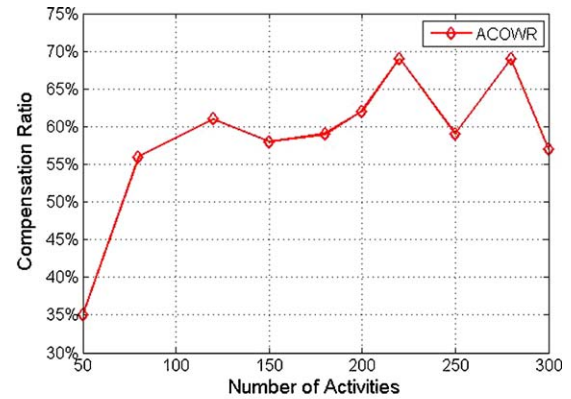
In this section, the experimental results for ACOWR are first demonstrated, followed by the simulation results for the evaluation of the example framework which includes TDA, ACOWR and TDA + ACOWR.

#### 5.3.1. Experimental results for ACOWR

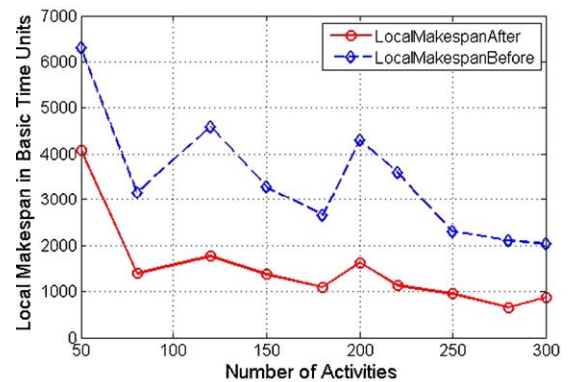
Given formula (1) in Definition 5, the compensation ratio is presented as follows in our experiments:

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

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. Since we want to investigate the capability of ACOWR in the handling of tem-



(a) The Compensation Ratio for Each Round of Experiment



(b) Local Makespan for Each Round of Experiment

Fig. 10. Compensation on violated workflow segment. (a) The compensation ratio for each round of experiment. (b) Local makespan for each round of experiment.

poral violations, we need to compare the local makespan of the violated workflow segment before and after the same rescheduling strategy in the same workflow instance. Here, the *LocalMakespanBefore* is defined as the mean execution time of the local workflow segment of all the solutions in the *SolutionSet* introduced in Section 4.2.2. Since the *SolutionSet* consists of all the best-so-far solutions in each iteration, the setting for the value of *LocalMakespanBefore* is fair to represent an average performance of scheduling plans.

Note that, in order to find out balanced solution with total makespan and cost, as presented in Fig. 7, the best-so-far solutions for the first searching stage are defined as those solutions with the largest fitness value in each iteration. After that, in the second searching stage, the final solution is selected as the one which can compensate the occurring time deficit while having the least cost among all the effective candidates.

The results of compensation on violated workflow segment are depicted in Fig. 10. Fig. 10(a) depicts the average compensation ratio for each round of experiment which is executed for 100 times. 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 large 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. Fig. 10(b) further illustrates the actual values of the average local makespan for the violated workflow segment before and after workflow rescheduling in each round of experiment. As shown in the figure, the two curves have similar behaviour while keeping a large distance all the time. Such a phenomenon actually denotes that ACOWR can compensate the

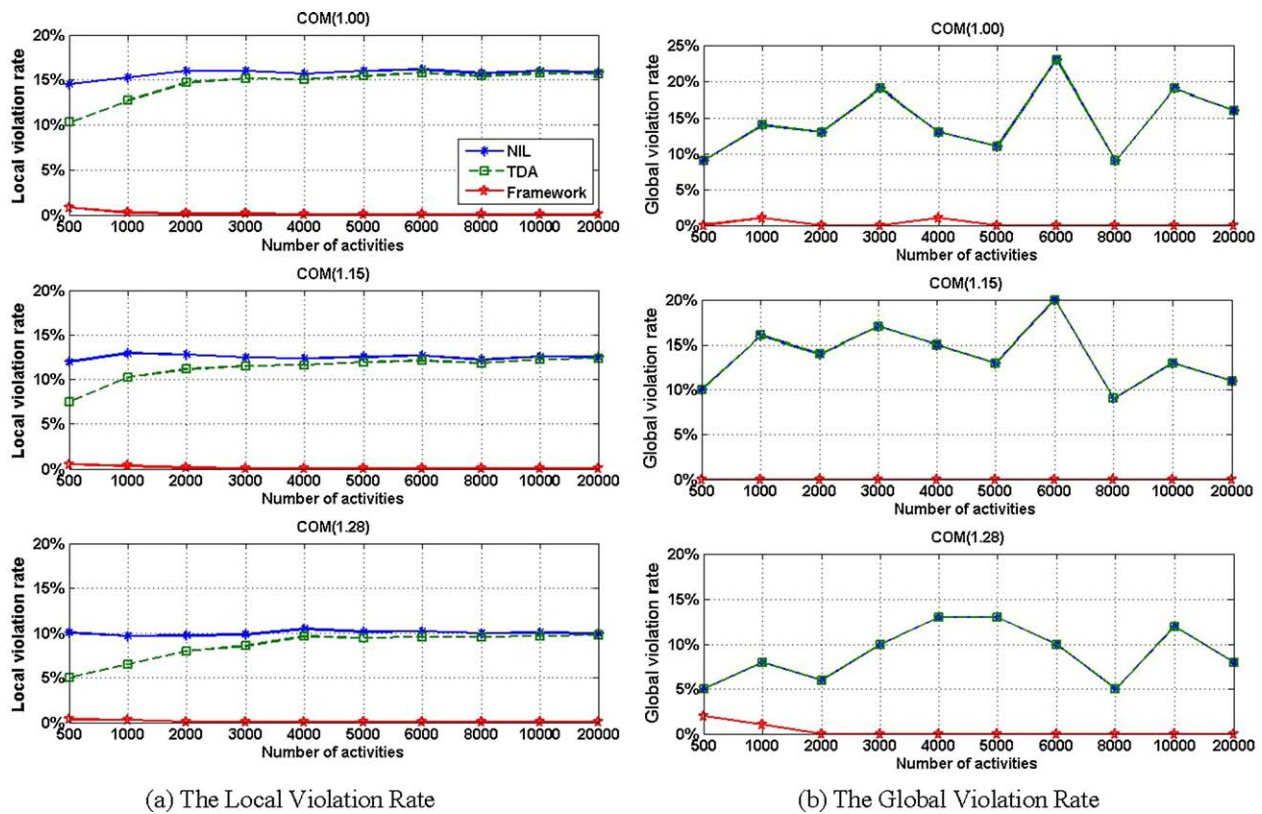


Fig. 11. Handling of temporal violations. (a) The local violation rate. (b) The global violation rate.

time deficit by effectively reducing the local workflow makespan of the violated workflow segment.

Furthermore, based on our additional experiments, ACOWR can effectively optimise the total makespan and total cost of the entire integrated task-resource list. 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 4.73 s which can be regarded as trivial compared with the durations of scientific workflow activities usually measured in minutes and hours. Therefore, we can claim that ACOWR is capable of compensating the violated workflow segment as well as optimising the entire integrated task-resource list on both total makespan and total cost. Therefore, the requirements for rescheduling strategies for handling temporal violations as analysed in Section 4.2.2 are met.

Our experimental results have also shown that other metaheuristic scheduling algorithms such as GA can also achieve similar compensation ratio but with different optimisation ratio on total makespan and total cost. Based on our observation, the reason for such a result is that given enough times of iteration, ACOWR and other metaheuristic scheduling algorithms can find the local-optimal for the local workflow segment of the violated workflow instance. The local-optimal is usually achieved when all the activities in the local workflow segment are mapped to the resources with highest execution speed. This local-optimal can often be found with the searching ability of metaheuristic scheduling algorithms. However, to balance between time deficit compensation and the completion time of other activities (i.e. the local-optimal and the global-optimal), the final solution is selected as the one which can compensate the occurring time deficit while having the least cost among all the valid solutions. Therefore, the final solution may not necessarily be the one with local-optimal but very close to that. For such a reason, there is usually a large space for reducing the makespan of the local workflow segment. Even when the

local workflow segment is mapped to the resource with high speed (i.e. there is little space for reducing the makespan), in such a case, ACOWR will be executed repeatedly by reading in more subsequent workflow segments to increase the searching space. Therefore, as the results demonstrated in the next section, the average compensation ratio is generally large enough for handling recoverable temporal violations.

### 5.3.2. Experimental results for example framework

As can be seen in Fig. 11(a), the local 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 4000. As analysed in Section 4.2.1, since TDA does not compensate time deficits, it cannot postpone 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 with our framework in each round are 0.16%, 0.13% and 0.09% respectively, i.e. an overall average of 0.127%. Fig. 11(b) shows the results on global violation rates. Since TDA has no effect on global violations, the global violation rates for NIL and TDA are overlapping. 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 example 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 example framework in each

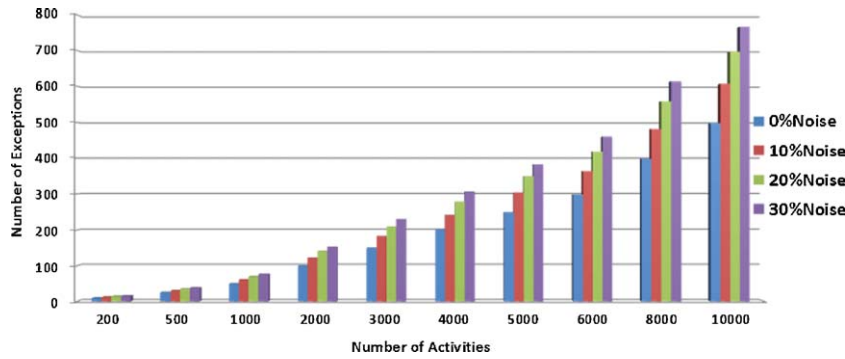


Fig. 12. Number of exceptions with different noises.

round are 0.2%, 0.0% and 0.3% respectively, i.e. an overall average of 0.167%.

Based on the above experimental results and according to formula (2) in Definition 5, we can see that the capability lower bound for TDA is around  $\theta\% - 5\%$ , the capability lower bound for ACOWR is  $\theta\% - 58.5\%$ , and the capability lower bound for TDA + ACOWR is  $\theta\% - 90\%$ . The reason can be explained as follows. For a standalone TDA, it can reduce the local violation rate by around 5%, but it will not be effective when the workflow size is large enough, e.g. over 1000. However, as in our example framework, TDA is only responsible for level I violations and it is applied together with ACOWR for larger violations, hence its capability bound is around 5% for dealing with local temporal violations since the size of a local workflow segment is usually much smaller than 1000. For ACOWR, its capability lower bound is defined according to its average *CompensationRatio* which is 58.5% given the experimental results shown in Fig. 10. As for TDA + ACOWR (where the maximum iteration times for ACOWR are 2), since the time deficits are randomly set as 50–90% of the mean durations of the local workflow segment (as in Table 1), and the average local violation rate for the three rounds of experiments are 0.13% (which is very close to 0%), the average *CompensationRatio* for TDA + ACOWR should be around or over 90% so as to maintain such a near 0% local violation rate. Therefore, the capability lower bound for TDA + ACOWR is defined as  $\theta\% - 90\%$ . Clearly, the simulation results are consistent with the definitions of the three levels of fine-grained temporal violations in our example framework as presented in Section 4.1.

### 5.3.3. Quantitative cost analysis for example framework

As discussed through qualitative analysis at the beginning of this section, our example framework is cost-effective. Here, we further present a quantitative cost analysis.

We first take a look at the number of exceptions (including all the three levels of temporal violations, and non-recoverable temporal violation if any) in scientific workflows. Given the same experimen-

tal settings as COM(1.28), 0% (i.e. no noise), 10%, 20% and 30% noises are added to simulate the system environments from smooth to non-smooth situations. Here, to add noise is to increase the durations of selected activities by 0%, 10%, 20% and 30% respectively. A random activity in each workflow segment with the average size of 10 will be selected as the noisy point. Fig. 12 depicts the number of exceptions with different noises under natural situations, i.e. without any handling strategies.

The number of exceptions increases accordingly with the growth of workflow size. It is also evident that when the noise is larger, the number of exceptions is also larger. For example, when the number of workflow activities is 6000, the numbers of exceptions are 298, 363, 419 and 460 for the noises of 0%, 10%, 20% and 30% respectively. When the workflow size increases, the differences are even larger. For example, when the number of workflow activities is 10,000, the numbers of exceptions are 498, 606, 696 and 766 for the noises of 0%, 10%, 20% and 30% respectively.

In our example framework, there are three exception handling strategies including TDA, ACOWR and TDA + ACOWR. As can be seen in Section 4.2, the cost of TDA is negligible compared with that of ACOWR. In such a case, the cost of TDA + ACOWR (including a maximum 2 times for ACOWR) can be regarded either the same or twice as that of ACOWR depending on how many times that ACOWR is facilitated. Therefore, to ease our discussion, we can use the equivalent number of times for the application of ACOWR to represent the cost (and/or overhead) given that the average CPU time for running ACOWR once in a SwinDeW-G node is 4.73 s as indicated in Section 5.3.1. Here, with various scientific workflows same as in Fig. 11, we compare the cost of a standalone ACOWR and the cost of standalone TDA + ACOWR with that of our example framework. Here, for the comparison purpose, the standalone ACOWR and standalone TDA + ACOWR are implemented separately as the sole strategies for handling all the three levels of temporal violations. The experiment settings are the same as in COM(1.28). Given similar global

Table 2

Experiment results on the times of exception handling.

Workflow size	Total times of exception handling	Total times of TDA	Total times of ACOWR	Total times of TDA + ACOWR	Equivalent times of ACOWR
Example framework					
2000	43.8	4.3	33.7	5.8 (3.5 (2), 2.3 (1))	43.0
5000	102.5	11.5	81.3	9.7 (6.2 (2), 3.5 (1))	97.2
10,000	195.8	23.5	157.2	15.1 (9.8 (2), 5.3 (1))	182.1
Standalone ACOWR					
2000	49.5	NIL	49.5	NIL	49.5
5000	123.8	NIL	123.8	NIL	123.8
10,000	248.6	NIL	248.8	NIL	248.8
Standalone TDA + ACOWR					
2000	46.3	NIL	NIL	463 (4.1 (2), 42.2 (1))	50.4
5000	118.5	NIL	NIL	118.5 (12.2 (2), 106.3 (1))	130.7
10,000	239.2	NIL	NIL	239.2 (21.5 (2), 217.7 (1))	260.7

violation rates, the results for our example framework and the two standalone strategies are shown in Table 2.

As discussed above, the total cost for each strategy is calculated as the equivalent times for the application of ACOWR. For example, as shown in Table 2, for our example framework with 2000 activities, on average, the total times of TDA are 4.3 (for level I temporal violation), the total times of ACOWR (for level II temporal violation) are 33.7, and the total times of TDA + ACOWR are 5.8 (for level III temporal violation, and non-recoverable temporal violation if any) where 3.5 times with ACOWR twice and 2.3 with ACOWR once. Therefore, in such a case, the equivalent times of ACOWR are calculated as  $33.7 + 3.5 \times 2 + 2.3 \times 1 = 43.0$ . The other scenarios can be calculated in the similar way. Meanwhile, we can see that with 10,000 activities, for the total of 195.8 times of exceptions, the ratio of level I, level II, and level III temporal violations (and non-recoverable temporal violation if any) are 12.0%, 80.3%, and 7.7% respectively. The experimental results show that our example framework has the smallest number of total exception handling among all the strategies where the equivalent times of ACOWR for the workflow sizes of 2000, 5000 and 10,000 are 43.0, 97.2 and 182.1 respectively. This is mainly because in our example framework, different levels of temporal violations will be handled by their corresponding exception handling strategies with the least cost. As for the standalone ACOWR and TDA + ACOWR, their equivalent times of ACOWR are actually very close to each other. The main reason is that in the standalone TDA + ACOWR, ACOWR will be executed for either once or twice according to the level of the occurring temporal violations, e.g. once for level I or level II, and maybe twice for level III. Therefore, the total cost of TDA + ACOWR, i.e. the total times for the application of ACOWR in TDA + ACOWR, are actually very close to that of the standalone ACOWR. Take the standalone TDA + ACOWR for example, the cost reductions by our example framework are 14.7%, 25.6% and 30.1% respectively for the workflow size of 2000, 5000 and 10,000. Therefore, we can claim that our example handling framework is cost-effective. Comprehensive results including the results for other workflows can be found in our online document (see footnote 1).

Note that we have also investigated several expensive heavy-weight exception handling strategies. Here, we take “Adding a New Resource” (“Add” in short) as an example, the time overhead for Add mainly consists of two parts, viz. the data transfer time for workflow activities and the set-up time for a new resource. Given similar data transfer time in ACOWR and Add due to the re-mapping of tasks and resources, the set-up time for a new resource<sup>3</sup> in our simulation environment is normally around several minutes, similar to that of a reserved resource in the Amazon Elastic Compute Cloud (EC2, <http://aws.amazon.com/ec2/>). Therefore, the time overhead for Add is very large and thus not suitable for handling temporal violations. Furthermore, the monetary cost for reserving and using a new resource is much higher in comparison to the computation cost required for ACOWR. As for the other representative strategies such as *Stop and Restart* (Cooper et al., 2004), *Processor Swapping* (Cooper et al., 2004) and *Workflow Restructure* (Russell et al., 2006b), they are either similar to Add or require human interventions. Therefore, given the basic requirements of *Automation* and *Cost-effectiveness* for handling recoverable temporal violations, all these expensive strategies are not suitable candidates. Detailed results have been included in our online document (see footnote 1).

### 5.3.4. Summary

To conclude, the experimental results demonstrate that our example framework has excellent performance in reducing both

local and global temporal violation rates. The three levels of temporal violations within the statistically recoverable range can be handled effectively by the corresponding three exception handling strategies. Since our example framework can be viewed as a representative case study for the general exception handling framework presented in Section 3, we can envisage that our general exception handling framework is automatic and cost-effective for handling temporal violations in scientific workflow systems.

## 6. Related work

Temporal constraint is one of the most important workflow QoS constraints besides cost, fidelity, reliability and security as discussed in Yu and Buyya (2005). In practice, a set of temporal constraints can be deemed as a QoS contract between clients and service providers. In order to successfully fulfil these contracts, efficient monitoring mechanisms such as checkpoint selection (Chen and Yang, in press) and temporal verification (Chen and Yang, 2007) are implemented to dynamically detect temporal violations. The work in Russell et al. (2006b) introduces five types of workflow exceptions where temporal violations can be classified into deadline expiry. The work in Russell et al. (2006a) proposes three alternate courses of recovery action which are 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 Chen and Yang (2007) proposes a time deficit allocation (TDA) strategy which compensates current time deficits by utilising the expected time redundancy of subsequent activities. Here, the time deficit is the amount of time that has been delayed given the actual durations and the temporal constraints while the time redundancy is the expected extra time between the mean durations and temporal constraints. Therefore, the actual role of TDA is to verify the possibility of auto-compensation by future workflow segments. However, since the time deficit is not truly reduced by TDA, this strategy can only postpone the violations of local constraints on some local workflow segments, but has no effectiveness on global constraints, e.g. the final deadlines. Therefore, in this paper, to handle both local and global temporal violations, those strategies which can indeed reduce the time deficits need to be investigated. 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 (Yu and Shi, 2007).

Workflow rescheduling, such as local rescheduling (which deals with the mapping of underlying resources to workflow activities within specific local workflow segments), is normally triggered by the violation of QoS constraints (Cooper et al., 2004). Workflow scheduling as well as workflow rescheduling are classical NP-complete problems (Choudhury et al., 2008). Therefore, many heuristic algorithms are proposed. The work in Yu and Buyya (2008) has presented a systematic overview of workflow scheduling algorithms for scientific grid computing. The major grid workflow scheduling algorithms have been classified into two basic categories which are best-effort based scheduling and QoS-constraint based scheduling. Best-effort based scheduling attempts to minimise the execution time with ignoring other factors such as cost while QoS-constraint based scheduling attempts to minimise performance under important QoS constraints, e.g. makespan minimisation under budget constraints or cost minimisation under deadline constraints. Many heuristic methods such as Minimum Completion Time, Min–Min, and Max–Min have been proposed

<sup>3</sup> Here, a new resource is a hot online machine in the system. The set-up time for a cold offline machine will normally be much longer.

for best-effort based scheduling (Tracy et al., 2001). As for QoS-constraint based scheduling, some metaheuristic methods such as GA (Genetic Algorithm) and SA (Simulated Annealing) have been proposed and exhibit satisfactory performance (Yu and Buyya, 2008). In recent years, Ant Colony Optimisation (ACO), 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 (Chen et al., 2007). The work in Chen and Zhang (2009) proposes an ant colony optimisation approach to address scientific workflow scheduling problems with various QoS requirements such as reliability constraints, makespan constraints and cost constraints. A balanced ACO algorithm for job scheduling is proposed in Chang et al. (2009) which can balance the entire system load while trying to minimise the makespan of a given set of jobs. Given the basic requirement of *Cost-effectiveness*, both time and cost need to be considered while time has a priority over cost since we focus more on reducing the time deficits during the compensation process. Therefore, QoS-constraint based scheduling algorithms are better choices for handling temporal violations. An ACO based local workflow rescheduling strategy is proposed in Liu et al. (2010) for handling temporal violations in scientific workflows.

Unfortunately, although there are many recent works on how to detect those fine-grained temporal violations, and some studies on how to handle temporal violations, the research on the design of an automatic and cost-effective exception handling framework which addresses both of the two aspects is still in its infancy.

## 7. Conclusions and future work

Latest studies in checkpoint selection and temporal verification can only detect temporal violations but cannot handle them. In this paper, the issue of handling temporal violations in scientific workflows has been systematically investigated and addressed by our proposed exception handling framework. Given the two fundamental requirements of *Automation* and *Cost-effectiveness*, a novel general exception handling framework has been proposed where fine-grained temporal violations are defined based on the empirical function for the capability lower bounds of exception handling strategies. As a representative case, an example exception handling framework which includes three levels of fine-grained temporal violations and their corresponding exception handling strategies TDA, ACOWR and TDA + ACOWR, has been presented. Its excellent performance in reducing both local and global temporal violation rates has been verified through comprehensive simulation experiments.

In the future, more scheduling algorithms such as those adopted by various workflow systems will be implemented to investigate and improve the performance of the exception handling framework in various system environments. Meanwhile, some heavy-weight exception handling strategies may also be investigated in order to handle those “non-recoverable” temporal violations outside the statistically recoverable range.

## Acknowledgments

This work is partially supported by Australian Research Council under Linkage Project LP0990393, the National Natural Science Foundation of China project under Grant No. 70871033. Part of this work, particularly the example framework, has been accepted by ICPADS'2010. We are also grateful for the discussions with Dr. W. van Straten and Ms. L. Levin from Swinburne Centre for Astrophysics and Supercomputing.

## References

- Buhr, P.A., Mok, W.Y.R., 2000. Advanced exception handling mechanisms. *IEEE Transactions on Software Engineering* 26 (9), 820–836.
- Buyya, R., Yeo, C.S., Venugopal, S., Broberg, J., Brandic, I., 2009. Cloud computing and emerging IT platforms: vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems* 25 (6), 599–616.
- Chang, R., Chang, J., Lin, P., 2009. An ant algorithm for balanced job scheduling in grids. *Future Generation Computer Systems* 25 (1), 20–27.
- Chen, J., Yang, Y., 2007. Multiple states based temporal consistency for dynamic verification of fixed-time constraints in grid workflow systems. *Concurrency and Computation: Practice and Experience* 19 (7), 965–982.
- Chen, J., Yang, Y., 2008. A taxonomy of grid workflow verification and validation. *Concurrency and Computation: Practice and Experience* 20 (4), 347–360.
- Chen, J., Yang, Y., in press. Temporal dependency based checkpoint selection for dynamic verification of temporal constraints in scientific workflow systems. *ACM Transactions on Software Engineering and Methodology*. <http://www.swinflow.org/papers/TOSEM.pdf> (accessed 01.09.10).
- Chen, J., Yang, Y., 2010. Localising temporal constraints in scientific workflows. *Journal of Computer and System Sciences* 76 (6), 464–474.
- Chen, W., Zhang, J., 2009. 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* 39 (1), 29–43.
- Chen, W., Zhang, J., Yu, Y., 2007. Workflow scheduling in grids: an ant colony optimization approach. In: *Proceedings of the 2007 IEEE Congress on Evolutionary Computation (CEC2007)*, Singapore, September 2007, pp. 3308–3315.
- Choudhury, P., Kumar, R., Chakrabarti, P.P., 2008. Hybrid scheduling of dynamic task graphs with selective duplication for multiprocessors under memory and time constraints. *IEEE Transactions on Parallel and Distributed Systems* 19 (7), 967–980.
- Cooper, K., Dasgupta, A., Kennedy, K., Koelbel, C., Mandal, A., 2004. New grid scheduling and rescheduling methods in the GrADS project. In: *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, Santa Fe, New Mexico, USA, April 2004, pp. 199–206.
- Deelman, E., Gannon, D., Shields, M., Taylor, I., 2008. Workflows and e-science: an overview of workflow system features and capabilities. *Future Generation Computer Systems* 25 (6), 528–540.
- Duan, H., Zeng, Q., Wang, H., Sun, S.X., Xu, D., 2009. Classification and evaluation of timed running schemas for workflow based on process mining. *Journal of Systems and Software* 82 (3), 400–410.
- Eder, J., Panagos, E., Rabinovich, M., 1999. Time constraints in workflow systems. In: *Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAISE99)*, Heidelberg, Germany, June 1999, pp. 286–300.
- Foster, I., Kesselman, C., 2004. *The Grid: Blueprint for a New Computing Infrastructure*, second ed. Morgan Kaufmann.
- Hagen, C., Alonso, G., 2000. Exception handling in workflow management systems. *IEEE Transactions on Software Engineering* 26 (10), 943–958.
- HPGC, 2009. *Proceedings of the Sixth High-Performance Grid Computing Workshop in conjunction with the 23rd Parallel and Distributed Processing Symposium (IPDPS09)*. IEEE, ISBN 978-1-4244-3750-4.
- Kim, K.H., Buyya, R., Kim, J., 2007. Power aware scheduling of bag-of-tasks applications with deadline constraints on DVS-enabled clusters. In: *Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 07)*, Rio de Janeiro, Brazil, May 2007, pp. 541–548.
- Law, A.M., Kelton, W.D., 2007. *Simulation Modelling and Analysis*, fourth ed. McGraw-Hill.
- Li, H., Yang, Y., Chen, T.Y., 2004. Resource constraints analysis of workflow specifications. *Journal of Systems and Software* 73 (2), 271–285.
- Liu, X., Chen, J., Liu, K., Yang, Y., 2008a. Forecasting duration intervals of scientific workflow activities based on time-series patterns. In: *Proceedings of the 4th IEEE International Conference on e-Science (e-Science08)*, Indianapolis, Indiana, USA, December 2008, pp. 23–30.
- Liu, X., Chen, J., Yang, Y., 2008b. A probabilistic strategy for setting temporal constraints in scientific workflows. In: *Proceedings of the 6th International Conference on Business Process Management (BPM08)*, Milan, Italy, September 2008, pp. 180–195.
- Liu, X., Chen, J., Wu, Z., Ni, Z., Yuan, D., Yang, Y., 2010. Handling recoverable temporal violations in scientific workflow systems: a workflow rescheduling based strategy. In: *Proceedings of the 10th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid10)*, Melbourne, Australia, May 2010, pp. 534–537.
- Liu, X., Ni, Z., Chen, J., Yang, Y., in press. A probabilistic strategy for temporal constraint management in scientific workflow systems. *Concurrency and Computation: Practice and Experience*. <http://www.ict.swin.edu.au/personal/xliu/doc/ConstraintManagement.pdf> (accessed 01.09.10).
- PDSEC, 2009. *Proceedings of the 10th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing, in conjunction with The 23rd Parallel and Distributed Processing Symposium (IPDPS09)*. IEEE, ISBN 978-1-4244-3750-4.
- Prodan, R., Fahringer, T., 2008. Overhead analysis of scientific workflows in grid environments. *IEEE Transactions on Parallel and Distributed Systems* 19 (March (3)), 378–393.
- Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., 2006a. Exception handling patterns in process-aware information systems. Technical Report BPM-06-04. Business Process Management (BPM) Center, Eindhoven University of Technology.

- Russell, N., van der Aalst, W.M.P., ter Hofstede, A.H.M., 2006b. Workflow exception patterns. In: *Proceedings of the 18th International Conference on Advanced Information Systems Engineering (CAiSE06)*, Berlin, Germany, June 2006, pp. 288–302.
- SECES, 2008. *Proceedings of the First International Workshop on Software Engineering for Computational Science and Engineering, in Conjunction with the 30th International Conference on Software Engineering (ICSE08)*, ACM, May 2008, ISBN 978-1-60558-079-1.
- Son, J.H., Kim, M.H., 2001. Improving the performance of time-constrained workflow processing. *Journal of Systems and Software* 58 (3), 211–219.
- Stroud, K.A., 2007. *Engineering Mathematics*, sixth ed. Palgrave Macmillan, New York.
- Taylor, I.J., Deelman, E., Gannon, D.B., Shields, M., 2007. *Workflows for e-Science: Scientific Workflows for Grids*. Springer.
- Tracy, D.B., Howard, J.S., Noah, B., 2001. Comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing* 61 (6), 810–837.
- van der Aalst, W.M.P., Hee, K.M.V., Reijers, H.A., 2000. Analysis of discrete-time stochastic petri nets. *Statistica Neerlandica* 54 (2), 237–255.
- Yang, Y., Liu, K., Chen, J., Lignier, J., Jin, H., 2007. Peer-to-peer based grid workflow runtime environment of SwinDeW-G. In: *Proceedings of the 3rd International Conference on e-Science and Grid Computing (e-Science07)*, Bangalore, India, December 2007, pp. 51–58.
- Yu, J., Buyya, R., 2005. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing* 3 (3), 171–200.
- Yu, J., Buyya, R., 2008. Workflow scheduling algorithms for grid computing, metaheuristics for scheduling in distributed computing environments. *Studies in Computational Intelligence*, Springer 146 (1), 173–214.
- Yu, Z., Shi, W., 2007. An adaptive rescheduling strategy for grid workflow applications. In: *Proceedings of the 2007 IEEE International Symposium on Parallel and Distributed Processing (IPDPS07)*, Long Beach, California USA, March 2007, pp. 115–122.
- Zeng, Q.T., Wang, H.Q., Xu, D.M., Duan, H., Han, Y.B., 2008. Conflict detection and resolution for workflows constrained by resources and non-determined durations. *Journal of Systems and Software* 81 (9), 1491–1504.
- Zhuge, H., Cheung, T., Pung, H., 2001. A timed workflow process model. *Journal of Systems and Software* 55 (3), 231–243.