# Do We Need to Handle Every Temporal Violation in Scientific Workflow Systems?

XIAO LIU, East China Normal University and Swinburne University of Technology
YUN YANG*, Anhui University and Swinburne University of Technology
DONG YUAN, Swinburne University of Technology
JINJUN CHEN, University of Technology Sydney

Scientific processes are usually time constrained with overall deadlines and local milestones. In scientific workflow systems, due to the dynamic nature of the underlying computing infrastructures such as grid and cloud, execution delays often take place and result in a large number of temporal violations. Since temporal violation handling is expensive in terms of both monetary costs and time overheads, an essential question aroused is that "do we need to handle every temporal violation in scientific workflow systems?". The answer would be "true" according to existing works on workflow temporal management which adopt the philosophy similar to the handling of functional exceptions, i.e. every temporal violation should be handled whenever it is detected. However, based on our observation, the phenomenon of self-recovery where execution delays can be automatically compensated for by the saved execution time of subsequent workflow activities has been entirely overlooked. Therefore, considering the non-functional nature of temporal violations, our answer is "not necessarily true". To take advantage of self-recovery, this paper proposes a novel adaptive temporal violation handling point selection strategy where this phenomenon is effectively utilised to avoid unnecessary temporal violation handling. Based on simulations of both real world scientific workflows and randomly generated test cases, the experimental results demonstrate that our strategy can significantly reduce the cost on temporal violation handling by over 96% while maintaining extreme low violation rate under normal circumstances.

Categories and Subject Descriptors: **D 2.4 [Software Engineering]** – Software/Program Verification, **D.2.5.f [Testing and Debugging]** – Error Handling and Recovery

General Terms: Algorithms, Performance, Reliability, Verification

Additional Key Words and Phrases: Scientific workflows, temporal constraints, temporal verification, violation handling point selection, quality of service

**ACM Reference Format**:

Xiao Liu, Yun Yang, Dong Yuan, Jinjun Chen, 201X. Do We Need to Handle Every Temporal Violation in Scientific Workflow Systems. *ACM Trans. Softw. Engin.□Method.*□X,□X,□Article□XX □(XXX

The research work reported in this paper is partly supported by Australian Research Council under LP0990393 and DP110101340, Natural Science Foundation of China under No. 61021004, and Shanghai Knowledge Service Platform for Trustworthy Internet of Things under No. ZF1213.

Author's addresses: Xiao Liu, Shanghai Key Laboratory of Trustworthy Computing, Software Engineering institute of East China Normal University, Shanghai, 200062, China & Faculty of Information and Communication Technologies, Swinburne University of Technology, Melbourne, Australia 3122; email: xliu@sei.ecnu.edu.cn; Yun Yang* (corresponding author), School of Computer Science and Technology, Anhui University, Hefei, 230039, China & Faculty of Information and Communication Technologies, Swinburne University of Technology, Melbourne, Australia 3122; email: yyang@swin.edu.au; Dong Yuan, Faculty of Information and Communication Technologies, Swinburne University of Technology, Melbourne, Australia 3122; email: dyuan@swin.edu.au; Jinjun Chen, Faculty of Engineering and Information Technology, University of Technology Sydney, Sydney, Australia 2007; email: jinjun.chen@uts.edu.au.

Permission to make digital or hardcopies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credits permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2010 ACM 1539-9087/2010/03-ART39 $15.00
DOI:http://dx.doi.org/10.1145/0000000.0000000

201X), 37 pages.
DOI:http://dx.doi.org/10.1145/0000000.0000000

## 1. INTRODUCTION

A scientific workflow system is a type of high-level middleware service for high performance computing infrastructures such as grid, peer-to-peer (p2p) and cloud computing [Buyya et al. 2009; Yang et al. 2007; Yu and Buyya 2007]. From the perspective of software engineering, a scientific workflow system is a type of scientific software in the area of Software Engineering for Computational Science and Engineering which is attracting increasing attention from software engineering [SECES 2008] and scientific computing research communities [PDSEC 2009]. It is responsible for modelling and executing large-scale processes in a variety of complex computation and data intensive applications such as astrophysics, climate modelling and earthquake simulation [Deelman et al. 2008; Wang et al. 2009]. One of the software engineering research issues in the design and development of scientific workflow systems is temporal verification. Generally speaking, the purpose of temporal verification is to detect and recover any temporal violations in scientific workflow build-time specifications and runtime executions [Chen and Yang 2007a; Chen and Yang 2011; Eder et al. 1999; Marjanovic and Orlowska 1999; Zhuge et al. 2001].

In reality, scientific workflows, and also some general software applications, normally require the completion before specific deadlines to achieve application goals on schedule. Otherwise, the usefulness of its execution results may be severely deteriorated. For example, a daily weather forecast scientific workflow has to be finished before the broadcasting of the weather forecast program everyday at, for instance, 6:00pm. For scientific research purposes, scientific workflows are usually deployed on distributed computing infrastructures such as grid/p2p/cloud with distributed geographical locations and/or highly dynamic performance [Buyya et al. 2009; Deelman et al. 2008; Yang et al. 2007]. Therefore, temporal violations, as a type of non-functional QoS (quality of service) exceptions, may often occur along scientific workflow execution. To ensure satisfactory temporal QoS, four basic tasks of workflow temporal management are currently investigated in scientific workflow systems: temporal constraint setting [Liu et al. 2011a], temporal violation checkpoint selection [Chen and Yang 2008b], temporal verification [Chen and Yang 2007a] and temporal violation handling [Liu et al. 2011c]. Among them, a runtime checkpoint selection strategy aims at selecting activity points to conduct temporal verification and/or violation handling for large scale scientific workflows which may contain thousands or even hundreds of thousands of activities [Taylor et al. 2007]. Temporal violation handling is to execute violation handling strategies which can compensate for the occurring time deficit (the time delays over specific temporal constraints) but would impose some additional cost. Generally speaking, the two fundamental requirements for delivering satisfactory temporal QoS in scientific workflow systems are *temporal conformance* and *cost effectiveness*.

*Temporal conformance*. The ultimate goal for workflow temporal management is to maintain satisfactory runtime temporal conformance, i.e. timely completion of scientific workflows according to assigned temporal constraints. Temporal conformance can be measured by the violation rate of temporal constraints, e.g. the violation rate of global deadlines (i.e. the number of scientific workflows failed with timely completion divided by the total number of scientific workflows). Evidently, the lower the violation rate, the better the temporal conformance is.

*Cost effectiveness*. Every task for workflow temporal management incurs some cost. Take a single temporal violation handling as an example, its cost can be primarily referred to monetary costs and time overheads of violation handling strategies which are normally non-trivial in scientific workflow systems [Prodan and Fahringer 2008]. The overall cost of temporal violation handling is proportional to the number of times that violation handling strategies have been executed, i.e. the number of selected temporal violation handling points (or handling points for short in this paper). Clearly, given similar temporal conformance (i.e. the temporal violation rate), the smaller the number of selected handling points, the better the cost effectiveness is.

The existing state-of-the-art work on workflow temporal management adopts the philosophy that to maintain satisfactory temporal conformance, similar to the handling of functional exceptions, temporal violation handling is triggered on every necessary and sufficient temporal checkpoint [Chen and Yang 2008b]. Here, necessity means that only those activity points with temporal violations are selected and sufficiency means that no activity points with temporal violations are omitted. This is regarded as the benchmark for the evaluation of checkpoint selection strategies. In the real world, the number of selected checkpoints increases rapidly with the size of scientific workflows [Chen and Yang 2011]. Therefore, the cost on handling temporal violations can be very high in large scale scientific workflows. In general, the cost is regarded worthwhile for the requirement of *temporal conformance*. But considering the other requirement of *cost effectiveness*, a problem arises naturally: "*do we need to handle every temporal violation in scientific workflow systems*?". Obviously, the answer would be "true" according to the existing works where necessity and sufficiency is the benchmark for checkpoint selection as well as violation handling point selection. However, in this paper, we argue that the answer is "not necessarily true" as there is a common phenomenon overlooked by most researchers that the execution delay, i.e. the occurring time deficit of a temporal violation detected at a checkpoint, may often be small enough so that the saved execution time of the subsequent workflow activities, i.e. the time redundancy (the redundant time between the execution time and the temporal constraint for the subsequent activities after the checkpoint), could automatically compensate for it. We name this phenomenon as "self-recovery". For example, if the occurring time deficit is 10 seconds, it will probably be compensated for by the time redundancy of the next activity which has a mean duration of 10 minutes. In conventional strategies, this 10-second time deficit must be treated with violation handling since a temporal violation is detected. Such unnecessary violation handling may impose a high cost eventually. Upon our observation (as to be illustrated by the motivating example in Section 2.1), self-recovery is common in scientific workflow systems where minor delays often occur due to highly dynamic performance of the underlying computing infrastructure. Therefore, there is a good chance to reduce the cost on handling temporal violations which deserves a systematic investigation.

In this paper, we aim at reducing the unnecessary violation handling cost for temporal violations as much as possible while maintaining satisfactory temporal conformance. Given the requirements of *temporal conformance* and *cost effectiveness*, the essence is to select *key* instead of *all* checkpoints as handling points to resolve temporal violations. In fact, conceptually, handling point selection is similar to software testing [Chen and Merkel 2008; Sommerville 2009] where the input domain is known but large (i.e. many checkpoints), hence only a small portion of the input domain is selected as test cases (i.e. handling points) so that software failures (i.e.

temporal violations) can be effectively detected and then handled with limited resources. Inspired by the idea of adaptive random testing [Chen and Merkel 2008], this paper proposes a novel adaptive temporal violation handling point selection strategy which would only select a small subset of the necessary and sufficient checkpoints as handling points. Differing from the existing work on checkpoint selection which utilises qualitative measurements, this paper employs a probability based runtime temporal consistency model to facilitate statistical analysis and measure temporal violations in a quantitative fashion. The probability of self-recovery and the adaptive probability threshold are the two basic parameters for selecting handling points in scientific workflow systems. Based on simulations of both real world scientific workflows and randomly generated test cases, the experimental results demonstrate that our strategy can significantly reduce the violation handling cost while maintaining satisfactory temporal conformance.

Specifically, this paper has made the following significant contributions with the details presented in the following sections:

1) For the first time, the problem of "temporal violation handling point selection" is identified.

2) For the first time, the phenomenon of "self-recovery" is formally defined.

3) For the first time, with our probability based temporal consistency model, an adaptive cost-effective temporal violation handling point selection strategy is proposed.

The remainder of the paper is organised as follows. Section 2 discusses a motivating example and analyses the problems. Section 3 presents an overview of the probability based runtime temporal consistency model. Section 4 proposes our novel adaptive violation handling point selection strategy. Section 5 demonstrates the comprehensive experimental results. Section 6 describes the related work. Finally, Section 7 addresses the conclusions and points out the future work.

## 2. MOTIVATING EXAMPLE AND PROBLEM ANALYSIS

### 2.1 Motivating Example

In this section, we present a motivating example in Astrophysics. Parkes Radio Telescope, one of the most famous radio telescopes in the world, is serving institutions from different nations (http://www.parkes.atnf.csiro.au/). Swinburne Astrophysics group (http://astronomy.swinburne.edu.au/) has been conducting pulsar searching surveys (http://astronomy.swin.edu.au/pulsar/) based on the observation data from Parkes Radio Telescope. The Parkes multibeam pulsar survey is very successful to date. The pulsar searching process is a typical scientific workflow which is collaborative, distributed, large scale, and it involves a great number of data intensive and computation intensive activities. For a single pulsar searching process, the average data volume (not including the raw stream data from the telescope) is over 4 terabytes and the average execution time is over 20 hours on Swinburne high performance supercomputing facility (http://astronomy.swinburne.edu.au/supercomputing/).

For the convenience of discussion, as depicted in Figure 1, we only illustrate some high-level workflow activities and focus on one path in the total of 13 parallel paths for different beams (the other parallel paths are of similar nature and denoted by cloud symbols). The average durations (normally with large variances) for high-level activities (those with sub-processes underneath) and three temporal constraints are also presented. Due to the resource sharing of the supercomputing facility, each pulsar searching workflow is normally required to be completed in one day. Therefore,

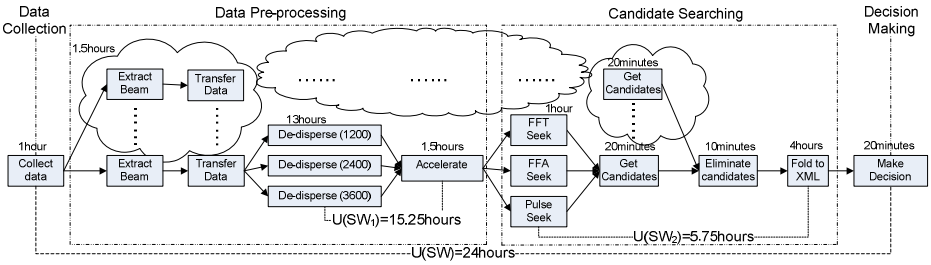



Fig. 1. An example scientific workflow for pulsar searching in astrophysics.

an overall upper bound temporal constraint $U(SW)$ of 24 hours, as the overall deadline, is assigned which denotes the acceptable maximum execution time.

Generally speaking, there are four main steps in the pulsar searching workflow, viz. data collection, data pre-processing, candidate searching and decision making. The first step is data collection (about 1 hour). Data from Parkes Radio Telescope streams at a rate of 1Gb per second and stored at local RAID (Redundant Array of Independent Disks). The second step is data pre-processing which consists of *Extract Beam*, *Transfer Data*, *De-disperse* and *Accelerate*. For data extraction and transfer (about 1.5 hours), different beam files are extracted and transferred via gigabyte optical fibre. The beam files contain the pulsar signals which are dispersed by the interstellar medium. De-dispersion is to counteract this effect. A large number of de-dispersion files are generated according to different choices of trial dispersions. In this scenario, 1,200 is the minimum number of dispersion trials, and it normally takes 13 hours to complete. For more dispersion trials such as 2,400 and 3,600, longer execution time is required or more computing resources need to be allocated. Furthermore, for binary pulsar searching, every de-dispersion file needs to undergo the *Accelerate* process. Each de-dispersion file generates several accelerated de-dispersion files and the whole process takes around 1.5 hours. For example, if the path with 1,200 de-dispersion files is chosen, then a temporal constraint of 15.25 hours, denoted as $U(SW_1)$, as one local milestone, can be assigned for the data pre-processing step. The third step is candidate searching which consists of *Seek*, *Get Candidates*, *Eliminate Candidates* and *Fold to XML*. Given the generated de-dispersion files, different seeking algorithms can be applied to search pulsar candidates, such as *FFT Seek*, *FFA Seek*, and *Pulse Seek*. For the instance of 1,200 de-dispersion files, it takes around 1 hour for *FFT Seek* to search all the candidates. Furthermore, by comparing the candidates generated from different beam files in the same time session (around 20 minutes), some interference may be detected and some candidates may be eliminated (around 10 minutes). With the final pulsar candidates, the original beam files are retrieved to find their feature signals and fold them to XML files. This activity usually takes around 4 hours. Here, a temporal constraint of 5.75 hours, denoted as $U(SW_2)$, as another local milestone, can be assigned for the candidate searching step. The last step is decision making where the XML files are inspected manually (by human experts) or automatically (by software) to facilitate the decision making on whether a possible pulsar is found or not (around 20 minutes).

Here, we only illustrate two representative situations to present the motivation of our work whilst in reality there could be many temporal constraints assigned for effective temporal verification at the lower levels. One situation is that during the second step of data pre-processing, delays may often occur in the activity of de-dispersion which needs to process terabytes of data and consumes more than 13

hours computation time. Here, for example, we assume that the de-dispersion activity takes 14.5 hours (a delay of 90 minutes, i.e. around 10% over the mean duration), then given $U(SW_1)$ of 15.25 hours, there would only be 45 minutes left for the *Accelerate* activity which normally needs 1.5 hours. Even if we expect it can be finished 10 minutes less than its mean duration (around 10% of the mean duration), i.e. the *Accelerate* activity can be finished in 80 minutes, there would still be a 35 minute time deficit. Therefore, under such a situation, at that stage, some violation handling should be invoked to decrease the duration for the *Accelerate* activity to at most 45 minutes by, for example, recruiting additional resources to maintain temporal conformance. Another situation is during the third step of pulsar seeking. We assume that the overall duration for the *FFT Seek*, *Get Candidates* and *Eliminate Candidates* activities is 108 minutes (a delay of 18 minutes, i.e. around 20% over the mean duration). In such a case, given $U(SW_2)$ being 5.75 hours, there will probably be a 3 minutes delay if the subsequent activity completes on time. However, this 3 minutes time deficit is a small fraction compared with the mean duration of 4 hours for the subsequent *Fold to XML* activity. Actually, there is a high probability that it will be automatically compensated for since it only requires the subsequent activity to be finished 1.25% shorter than its mean duration. In such a situation, it is normally unnecessary to trigger violation handling. However, regardless the probability of self-recovery, violation handling is always invoked in conventional strategies since a temporal violation is in fact detected.

Based on the observation, with a large number of individual workflow activities, the situations similar to the second one described above often occur, especially when the underlying resources are of highly dynamic performance. Therefore, if we can define and detect such a phenomenon, there is a good chance of significantly reducing the number, hence the cost, of temporal violation handling.

### 2.2 Problem Analysis

In the existing work on workflow temporal management [Chen and Yang 2008b; Chen and Yang 2011], temporal violation handling point selection is not regarded as an independent task since they normally adopt the philosophy that temporal violation handling should be conducted whenever a temporal violation is detected, i.e. handling point selection is the same as checkpoint selection by nature. Accordingly, no matter whether it is a major time deficit of 35 minutes or a minor time deficit of 3 minutes as described in the motivating example, it is selected as a handling point and followed by applying temporal violation handling strategies. However, in the real world, it is normally unnecessary to trigger violation handling for a minor time deficit since there is a high probability that it can be automatically compensated for by the time redundancy of the subsequent activities as the second situation described in the motivating example. Therefore, a checkpoint is not necessarily a handling point.

As indicated earlier, given the two fundamental requirements of *temporal conformance* and *cost effectiveness*, the challenging issue for temporal violation handling point selection is "*how to select the key checkpoints*". To address such an issue, we need to solve the following two major problems.

1) *How to measure temporal violations in a quantitative fashion*

To reduce the overall cost of temporal violation handling is to reduce the total number of times that temporal violation handling strategies have been invoked. For such a purpose, we need to detect those "minor" temporal violations which have relatively small time deficits. Therefore, it is important that we are able to measure

temporal violations in a quantitative fashion. However, most of the existing temporal consistency models utilise static time attributes such as the maximum, mean and minimum durations to define coarse-grained qualitative expressions such as the violation of strong consistency, weak consistency, weak inconsistency and strong inconsistency to measure the occurring temporal violations [Chen and Yang 2007b; Chen and Yang 2011]. To facilitate the statistical analysis of the occurring time deficit and the expected time redundancy at workflow runtime, it is better to model activity durations with dynamic variables (following probability distribution models) instead of static time attributes, especially in dynamic system environments [Law and Kelton 2007; Liu et al. 2011a]. Therefore, quantitative measurement of temporal violations is required based on statistic models.

2) *How to decide whether a checkpoint needs to be selected as a handling point or not*

To reduce the number of handling points is to omit those necessary and sufficient checkpoints where the occurring minor time deficits have a high probability of being compensated for by the expected time redundancy of subsequent activities, i.e. self-recovery. For a necessary and sufficient checkpoint where a temporal violation is detected, the occurring time deficit detected at the checkpoint and the expected time redundancy of the subsequent activities after the checkpoint are the basic factors to decide whether a checkpoint should be selected as a handling point or not. Therefore, an effective strategy is required to define the time deficit and time redundancy, and estimate the probability of self-recovery so as to determine the selection of temporal violation handling points in scientific workflow systems.

## 3. PRELIMINARY: A PROBABILITY BASED RUNTIME TEMPORAL CONSISTENCY MODEL

In this section, we present an overview of a probability based runtime temporal consistency model which is the preliminary for our adaptive temporal violation handling point selection strategy to be proposed in Section 4. Conventional discrete-state based temporal consistency models are of very limited ability in performing complex statistical analysis. Therefore, as proposed in [Liu et al. 2008; Liu et al. 2011c], a probability based temporal consistency model is defined to estimate the probability of meeting given temporal constraints. This model employs the popular "$3\sigma$" rule to specify activity durations [Stroud 2007]. The "$3\sigma$" rule depicts that for any sample coming from a normal distribution model, it has a probability of 99.74% to fall into the range of $[\mu-3\sigma,\mu+3\sigma]$, i.e. the probability range of (0.13%, 99.87%), where $\mu$ is the expected value and $\sigma$ is the standard deviation. Therefore, the maximum, mean and minimum durations of activity $a_i$ can be 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 the sample mean and $\sigma_i$ is the sample standard deviation. Its actual duration at runtime is denoted as $R(a_i)$. The expected value and standard deviation can be obtained from historic data such as scientific workflow system logs through statistical methods [Law and Kelton 2007]. Based on the approaches proposed in [Liu et al. 2011a; Liu et al. 2011c], the workflow completion time can be effectively estimated with the joint normal distribution of individual activity durations.

Generally speaking, temporal constraints mainly include three types: upper bound, lower bound and fixed-time [Chen and Yang 2007b]. 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. A lower bound constraint between two activities is a relative time value so that the duration between them must be greater

or equal to it. A fixed-time constraint at an activity is an absolute time value by which the activity must be completed. As discussed in [Chen and Yang 2007a], conceptually, a lower bound constraint is symmetrical to an upper bound constraint, and a fixed-time constraint can be regarded as a special case of an upper bound constraint of which the start time of the workflow is determined. Hence upper bound is the most general type of temporal constraints. Therefore, we only focus on upper bound constraints in this paper.

The probability based runtime temporal consistency model is defined as follows.

***Definition 1 (Temporal Consistency Model)***: For a scientific workflow $SW$ which covers activity $a_l$ to activity $a_n$, at activity $a_p$ ($p \le n$), the upper bound temporal constraint $U(SW)$ with the value of $u(a_l, a_n)$, is said to be of:

1) *Absolute Consistency* (AC),

   if $R(a_l, a_p) + \sum_{i=p+1}^{n} (\mu_i + 3\sigma_i) < u(a_l, a_n)$;

2) *Absolute Inconsistency* (AI),

   if $R(a_l, a_p) + \sum_{i=p+1}^{n} (\mu_i - 3\sigma_i) > u(a_l, a_n)$;

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

   if $R(a_l, a_p) + \sum_{i=p+1}^{n} (\mu_i + \lambda\sigma_i) = u(a_l, a_n)$.

Here, $U(SW)$ with the value of $u(a_l, a_n)$ denotes an upper bound temporal constraint which covers the activities from $a_l$ to $a_n$, $R(a_l, a_p)$ denotes the sum of runtime durations, $\lambda$ ($-3 \le \lambda \le 3$) is defined as the $\alpha$% ($0 < \alpha < 100$) confidence percentile with $F(\mu_i + \lambda\sigma_i) = \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{\mu_i + \lambda\sigma_i} e^{-(x-\mu_i)^2 / 2\sigma_i^2} \bullet dx = \alpha\%$ (the cumulative normal distribution function) [Stroud 2007]. For $R(a_l, a_p) + \sum_{i=p+1}^{n} (\mu_i + \lambda\sigma_i) \ge u(a_l, a_n)$, we state that $U(SW)$ is above $\alpha$% *Consistency*.

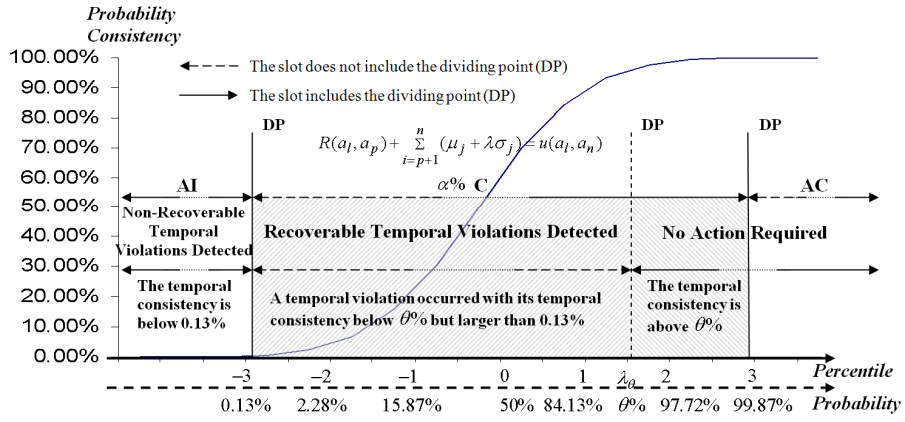



Fig. 2. Probability based temporal consistency model.

Note that for the convenience of discussion, in this paper, we only consider a single execution path in DAG (directed acyclic graph) based workflow instance [Chen and Yang 2008b], but the results can be applied equally to multiple paths (e.g. parallel structures) similarly by the repetition method as used in [Chen and Yang 2007a; Chen and Yang 2008b]. Specifically, 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.

As depicted in Figure 2, every temporal consistency state is represented with a unique probability value and they jointly form a continuous Gaussian curve which stands for the cumulative normal distribution [Stroud 2007]. According to the "$3\sigma$" rule, most runtime temporal consistency states (i.e. 99.74% of them) belong to the probability consistency range of (0.13%, 99.87%) which is represented by the shadowed area. At scientific workflow runtime, a temporal violation occurs when the probability consistency is below $\theta$% as shown in the area marked with upwards diagonal lines in Figure 2, while the probability consistency range of ($\theta$%,99.87%) marked with downwards diagonal lines requires no action. Here, the threshold of $\theta$% denotes the minimum acceptable temporal consistency state and it is usually specified through the negotiation between users and service providers as a type of QoS contract. In practice, $\theta$% would normally be around 90%, i.e. $\mu+1.28\sigma$, to serve as a satisfactory user requirement on temporal conformance [Liu et al. 2011a]. Therefore, if temporal consistency of $\alpha$% ($\alpha$% *Consistency*) is larger than $\theta$%, including AC (*Absolute Consistency*), no action is required as the QoS contract holds. Otherwise, if $\alpha$% is smaller than $\theta$%, including AI (*Absolute Inconsistency*), violation handling should be triggered for temporal violations according to the conventional philosophy of workflow temporal management since a temporal violation is detected. However, this will no longer be the case given our contribution on the temporal violation handling point selection proposed in this paper. The details will be presented in Section 4. Meanwhile, note that AI ($\alpha$% is smaller than 0.13%) is normally regarded as non-recoverable unless additional computing resources are recruited from the outside of the current system [Liu et al. 2011b]. In this paper, we focus on recoverable temporal violations within the range of (0.13%, $\theta$%). The work in [Hagen and Alonso 2000; Russell et al. 2006a] can be referred for a general overview on exception handling in workflow systems.

## 4. NOVEL ADAPTIVE TEMPORAL VIOLATION HANDLING POINT SELECTION STRATEGY

Motivated by the phenomenon of self-recovery and the idea of adaptive random testing, our novel adaptive temporal violation handling point selection strategy aims at selecting a small subset of the necessary and sufficient temporal checkpoints as handling points. The state-of-the-art checkpoint selection strategy is the temporal dependency based checkpoint selection strategy ($CSS_{TD}$) which is proposed in [Chen and Yang 2011]. Therefore, we use the results of $CSS_{TD}$ as the input for our strategy. But since $CSS_{TD}$ employs the multiple states based temporal consistency model [Chen and Yang 2007b] to define the temporal dependency, we need to revise the definition to match our probability based temporal consistency model. Therefore, in

Section 4.1, we first review and revise the definition of temporal dependency, and then the two basic theorems for $CSS_{TD}$ are modified accordingly and proved to demonstrate that our temporal violation handling point selection strategy can be seamlessly integrated with $CSS_{TD}$ . Afterwards, in Section 4.2, we present our adaptive temporal violation handling point selection strategy based on the checkpoint selection results of $CSS_{TD}$ .

### 4.1 Probability Temporal Dependency

Scientific workflows are normally of large size and may consist of many sub-processes, hence scientific workflow activities are often covered by multiple temporal constraints such as coarse-grained temporal constraints (e.g. global deadlines) and fine-grained temporal constraints (e.g. local milestones) [Liu et al. 2011a]. Therefore, multiple times of temporal verification are often required for all temporal constraints to detect temporal violations and their time deficits at all checkpoints. Unfortunately, this will give rise to a huge increase in the computation cost and time overheads for temporal verification. To address such a problem, $CSS_{TD}$ investigates the temporal dependency between different upper bound temporal constraints. The results have shown that at any specific checkpoint, for those temporal constraints which cover the checkpoint and have the same type of temporal dependency, such as SC (Strong Consistency) temporal dependency, temporal verification is needed once only to verify all of them. Therefore, many times of unnecessary temporal verification are avoided. Based on the similar idea, in this paper, we apply temporal dependency in our temporal violation handling point selection strategy for multiple temporal constraints.

Fig. 3. Two nested upper bound temporal constraints.

The work in [Chen and Yang 2011] has introduced both non-nested and nested scenarios for temporal constraints based on Allen's temporal interval logic [Allen 1983]. Since non-nested temporal constraints have no temporal dependency, we only focus on nested scenarios in this paper. Here, we start from the illustration of two nested upper bound temporal constraints as depicted in Figure 3 and then move on to more general cases with a series of upper bound temporal constraints as depicted in Figure 4.

As depicted in Figure 3, in general, there are three types of temporal dependency between two nested temporal constraints, viz. general nested, left joint and right joint. We start from Scenario 1. Here, we have two upper bound constraints $U_1$ and $U_2$ , and we assume they have $\theta\%$ probability temporal dependency and $a_p$ ( $i_2 < i_1 < p < j_1 < j_2$ ) is selected as the checkpoint for $U_1$ and $U_2$ by $CSS_{TD}$ . Here, similar to SC temporal consistency as defined in [Chen and Yang 2011], the $\theta\%$ probability temporal dependency is defined as follows.

***Definition 2**: (θ% Probability Temporal Dependency).*

Given $U_1$ and $U_2$ illustrated above where $U_1$ is between $a_{i_1}$ and $a_{j_1}$ and $U_2$ is between $a_{i_2}$ and $a_{j_2}$ ($i_2 < i_1 < j_1 < j_2$), namely $U_1$ is nested in $U_2$ as in Scenario 1, then, if $\theta(a_{i_2}, a_{i_1-1}) + u(a_{i_1}, a_{j_1}) + \theta(a_{j_1+1}, a_{j_2}) \le u(a_{i_2}, a_{j_2})$, $\theta$% probability temporal dependency between $U_1$ and $U_2$ is defined as consistent, or in other words, $U_1$ and $U_2$ have $\theta$% probability temporal dependency. Here, $\theta(a_{i_2}, a_{i_1-1}) = \sum_{k=i_2}^{i_1-1}(\mu_i + \lambda_\theta \sigma_i)$ where $\lambda_\theta$ is the $\theta$% confidence percentile as explained in **Definition 1**, and others are similar.

**Definition 2** also applies to Scenario 2 and Scenario 3 where $i_2 \le i_1 < j_1 \le j_2$.

We now investigate the probability temporal dependency between more general cases with a series of temporal constraints. As depicted in Figure 4, we can see that Scenario 4, Scenario 5 and Scenario 6 are extensions of Scenario 1, Scenario 2 and Scenario 3 respectively. Scenario 7 is the combination of Scenario 4, Scenario 5 and Scenario 6.

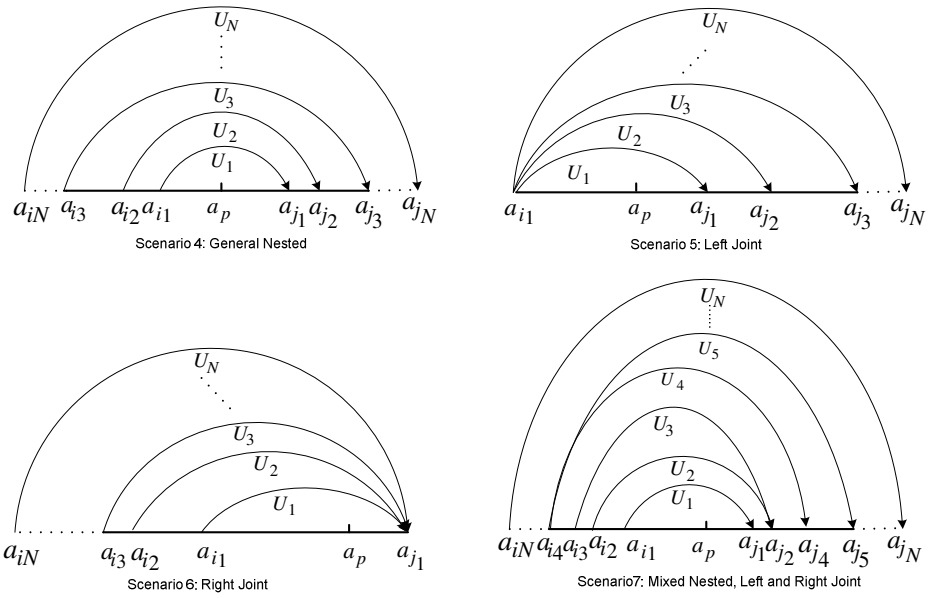



Fig. 4. A series of nested upper bound temporal constraints.

The core of $CSS_{TD}$ is based on two theorems. The first theorem states that the temporal dependency is transitive. The second theorem states that for temporal constraints with temporal dependency, if the previous temporal constraints (e.g. $U_2$) are consistent, then the later temporal constraints (e.g. $U_3$) must be consistent. Here, we modify these two theorems according to our probability based temporal consistency model and prove that the conclusion is the same.

THEOREM 1. Let $U_1$, $U_2$,.., $U_N$ be the $N$ upper bound constraints (as in Figure 4) where $U_1$ is between $a_{i_1}$ and $a_{j_1}$, and $U_2$ is between $a_{i_2}$ and $a_{j_2}$ and so forth

($i_N \le ... \le i_2 \le i_1 < j_1 \le j_2 \le ... \le j_N$), namely $U_1$ is nested in $U_2$, $U_2$ is nested in $U_3$ and so forth. Then, if probability temporal dependency between two adjacent upper bound constraints $U_k$ and $U_{k+1}$ is consistent ($1 \le k \le N-1$), the probability temporal dependency between any two non-adjacent upper bound constraints must also be consistent.

PROOF: Since Scenarios 5, 6, and 7 can actually be viewed as special cases of Scenario 4, we take Scenario 4 as an example to conduct the proof. For simplicity, we consider $U_1$, $U_2$, and $U_3$. Suppose $\theta$% temporal dependency between $U_1$ and $U_2$ is consistent and $\theta$% temporal dependency between $U_2$ and $U_3$ is also consistent. Now we prove that $\theta$% temporal dependency between $U_1$ and $U_3$ is also consistent. That is to say, the probability temporal dependency is transitive. According to **Definition 2**, we have inequations (1) and (2) below:

$$\theta(a_{i_2}, a_{i_1-1}) + u(a_{i_1}, a_{j_1}) + \theta(a_{j_1+1}, a_{j_2}) \le u(a_{i_2}, a_{j_2}) \quad (1)$$

$$\theta(a_{i_3}, a_{i_2-1}) + u(a_{i_2}, a_{j_2}) + \theta(a_{j_2+1}, a_{j_3}) \le u(a_{i_3}, a_{j_3}) \quad (2)$$

Given inequations (1) and (2), we can have $\theta(a_{i_3}, a_{i_1-1}) + u(a_{i_1}, a_{j_1}) + \theta(a_{j_1+1}, a_{j_3}) =$ $\theta(a_{i_3}, a_{i_2-1}) + \theta(a_{i_2}, a_{i_1-1}) + u(a_{i_1}, a_{j_1}) + \theta(a_{j_1+1}, a_{j_2}) + \theta(a_{j_2+1}, a_{j_3})$ $\le \theta(a_{i_3}, a_{i_2-1}) + u(a_{i_2}, a_{j_2}) + \theta(a_{j_2+1}, a_{j_3}) \le u(a_{i_3}, a_{j_3})$, hence we can have inequation (3) below:

$$\theta(a_{i_3}, a_{i_1-1}) + u(a_{i_1}, a_{j_1}) + \theta(a_{j_1+1}, a_{j_3}) \le u(a_{i_3}, a_{j_3}) \quad (3)$$

According to **Definition 2,** inequation (3) means that the $\theta$% probability temporal dependency between $U_1$ and $U_3$ is also consistent. Thus the theorem holds.

THEOREM 1 is used at scientific workflow build time to verify the probability temporal dependency between nested upper bound constraints. Afterwards, at runtime execution stage, we can use THEOREM 2 which is defined as follows to deduce the consistency of later constraints (e.g. $U_3$) from previous ones (e.g. $U_2$).

THEOREM 2**:** Consider two upper bound constraints $U_k$ and $U_s$ ($k < s \le N$) where $\theta$% probability temporal dependency between $U_k$ and $U_s$ is consistent; $U_k$ is between $a_{i_k}$ and $a_{j_k}$, and $U_s$ is between $a_{i_s}$ and $a_{j_s}$ and ($i_s < i_k < j_k < j_s$), namely $U_k$ is nested in $U_s$. Then, at checkpoint $a_p$ between $a_{i_k}$ and $a_{j_k}$, if $R(a_{i_s}, a_{i_k-1}) \le \theta(a_{i_s}, a_{i_k-1})$ and $U_k$ is of $\theta$% consistency, then $U_s$ must also be at least of $\theta$% consistency.

PROOF: if $U_k$ is of $\theta$% consistency, then according to **Definition 1**, we have inequation (4) below:

$$R(a_{i_k}, a_p) + \theta(a_{i_{p+1}}, a_{j_k}) = u(a_{i_k}, a_{j_k}) \quad (4)$$

Meanwhile, since $\theta$% probability temporal dependency between $U_k$ and $U_s$ is consistent, according to **Definition 2**, we have inequation (5) below:

$$\theta(a_{i_s}, a_{i_k-1}) + u(a_{i_k}, a_{j_k}) + \theta(a_{j_k+1}, a_{j_s}) \le u(a_{i_s}, a_{j_s}) \quad (5)$$

if $R(a_{i_s}, a_{i_k-1}) \le \theta(a_{i_s}, a_{i_k-1})$, then based on inequations (4) and (5), we can have $\theta(a_{i_s}, a_{i_k-1}) + R(a_{i_k}, a_p) + \theta(a_{i_{p+1}}, a_{j_k}) + \theta(a_{j_k+1}, a_{j_s}) \le u(a_{i_s}, a_{j_s})$, and then $R(a_{i_s}, a_{i_k-1}) + R(a_{i_k}, a_p) + \theta(a_{i_{p+1}}, a_{j_k}) + \theta(a_{j_k+1}, a_{j_s}) \le u(a_{i_s}, a_{j_s})$, and finally we can

have $R(a_{i_s}, a_{i_p}) + \theta(a_{i_{p+1}}, a_{j_s}) \leq u(a_{i_s}, a_{j_s})$ . According to **Definition 1**, this result means that $U_s$ is of at least $\theta\%$ consistency.

Based on THEOREM 1 and THEOREM 2, we can conclude that the results obtained by $CSS_{TD}$ with the multiple states based temporal consistency model can also be applied to our probability based temporal consistency model. The detailed process of $CSS_{TD}$ can be found in [Chen and Yang 2008b; Chen and Yang 2011] and hence omitted here. In our violation handling point selection strategy, we use the checkpoint selection results of $CSS_{TD}$ as the input, so that the quality of the selected checkpoints is guaranteed.

### 4.2 Novel Handling Point Selection Strategy

With the probability based runtime temporal consistency model, we can quantitatively measure different temporal violations based on their probability temporal consistency states. Furthermore, at a specific checkpoint, the occurring time deficit and the expected time redundancy of subsequent activities need to be defined. We start from the definition of the probability time deficit.

***Definition 3:*** *(Probability Time Deficit).*

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

**Definition 3** defines the probability time deficit at checkpoint $a_p$ for a single temporal constraint $U(SW)$. As discussed above, multiple temporal constraints are common in scientific workflows. Therefore, the probability time deficit for multiple temporal constraints should be considered. Based on **Definition 3**, we have defined the maximum probability time deficit as follows.

***Definition 4:*** *(Maximum Probability Time Deficit).*

Let $U_1$, $U_2$,.., $U_N$ be the $N$ upper bound 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 $U_1$, $U_2$ ,.., $U_N$ and is represented as $MPTD(a_p) = Max\{PTD(U_s, a_p) | s = 1,2,...,N\}$.

The purpose of **Definition 4** is to consider the general scenario with multiple temporal constraints like in Figure 4. Clearly, to handle temporal violations, we need to compensate for the maximum occurring probability time deficit, so that all the temporal violations can be recovered. Next, we define the probability time redundancy.

***Definition 5****: (Probability Time Redundancy).*

At activity point $a_p$, let $U(SW)$ be of $\beta\%$ C with the percentile of $\lambda_\beta$ which is above the threshold of $\theta\%$ with the percentile of $\lambda_\theta$. The subsequent activities are defined as those activities from the next activity of the checkpoint, i.e. $a_{p+1}$, to the

end activity of the next temporal constraint, e.g. $a_{p+m}$. With a segment of size $m$ from $a_{p+1}$ to $a_{p+m}$, the probability time redundancy of subsequent activities is $PTR(U(SW),(a_{p+1},a_{p+m}))$ which is equal to $u(a_l,a_n)-[R(a_1,a_p)+M(a_{p+1},a_{p+m})+\theta(a_{p+m+1},a_n)]$. Here, $M(a_{p+1},a_{p+m})$ is equal to $\sum_{k=p+1}^{p+m}(\mu_k)$ and $\theta(a_{p+m+1},a_n)$ is equal to $\sum_{k=p+m+1}^{n}(\mu_k+\lambda_\theta\sigma_k)$.

Similarly, we define the minimum probability time redundancy considering the general scenario with multiple temporal constraints. Clearly, when temporal violations are detected, the minimum probability time redundancy is preferred to be used to compensate for the time deficit in such a tentative situation. Otherwise, if the action fails, namely the skipping of the checkpoint as a handling point is unsuccessful, the subsequent temporal violations will become more serious.

***Definition 6:*** *(Minimum Probability Time Redundancy).*

Let $U_1$, $U_2$,.., $U_N$ be the $N$ upper bound constraints and all of them cover $a_p$, then, at $a_p$, the minimum probability time redundancy is defined as the minimum of all probability time redundancies of $U_1$, $U_2$,.., $U_N$ and is represented as $MPTR(a_p) = Min\{PTR(U_s,(a_{p+1},a_{p+m}))|s=1,2,...,N\}$.

The maximum probability time deficit is for measuring the occurring time deficit at the current checkpoint. The minimum probability time redundancy is for measuring the expected time redundancy (i.e. the time redundancy between the mean durations and the temporal constraints) of the subsequent activities at the current checkpoint. For example, at checkpoint $a_p$, if the temporal constraint for activity $a_{p+1}$ to $a_{p+m}$ is equal to $\mu+\lambda_\theta\sigma$ where $\mu$ and $\sigma$ are the joint normal mean and standard deviation respectively, then the value of $\lambda_\theta\sigma$ is regarded as the expected time redundancy which can be used to compensate for the occurring time deficit. Furthermore, based on **Definition 4** and **Definition 6** above, the probability of self-recovery is defined as follows.

***Definition 7:*** *(Probability of Self-Recovery).*

For activity point $a_p$ which is covered by $U_1$, $U_2$,.., $U_N$, given the maximum probability time deficit (denoted as $MPTD(a_p)$) and the minimum probability time redundancy (denoted as $MPTR(a_p)$), the probability of self-recovery, i.e. the probability that $MPTD(a_p)$ can be compensated for by $MPTR(a_p)$ is defined as:

$$P(T)=\frac{1}{\sqrt{2\pi}}\int_{-\infty}^{T}e^{\frac{-x^2}{2}}, \text{ where } T=\frac{MPTR(a_p)-MPTD(a_p)}{MPTD(a_p)}$$

Here, without any prior knowledge, it is difficult to decide which probability distribution model that $T$ fits. Therefore, in this paper, we assume that $T$ follows a

standard normal distribution[1], i.e. $N(0,1)$ with the expected value of 0 and the standard deviation of 1. It is obvious that the larger the difference between $MPTR(a_p)$ and $MPTD(a_p)$, the higher the probability for self-recovery. For example, according to **Definition 5**, if $MPTR(a_p)$ is equal to $MPTD(a_p)$, i.e. $T$ is equal to 0, the probability for self-recovery is 50%. If $MPTR(a_p)$ is twice as large as $MPTD(a_p)$, i.e. $T$ is equal to 1, the probability for self-recovery is 84.13% [Stroud 2007]. Note that in practice, historic data can be employed to discover and modify the actual probability distribution models. Nevertheless, the strategy proposed here can be applied in a similar manner.

After the probability of self-recovery is defined, a probability threshold is required. The probability threshold can be regarded as the minimum confidence for skipping temporal violation handling on a selected checkpoint yet still retaining satisfactory temporal conformance. Given a probability threshold, a temporal violation handling point selection rule is defined as follows.

***Definition 8: (****Temporal Violation Handling Point Selection Rule).*

At activity $a_p$, with the probability of self-recovery $P(T)$ and the probability threshold $PT$ $(0 < PT < 1)$, the rule for temporal violation handling point selection is as follows: if $P(T) > PT$, then the current checkpoint is not selected as a handling point; otherwise, the current checkpoint is selected as a handling point.

The probability threshold $PT$ is an important parameter in the handling point selection rule and it needs to be defined to facilitate handling point selection at workflow runtime. However, whether self-recovery will be successful or not can only be determined after the execution of the subsequent activities. It is difficult, if not impossible, to specify the optimal probability threshold which can select the minimal number of handling points while maintaining satisfactory temporal conformance. To address this problem, we borrow the idea from adaptive random testing where new test cases are generated based on the knowledge of previous test cases [Chen and Merkel 2008]. In our strategy, the probability threshold can start from any moderate initial values, e.g. 0.5, and it is then adaptively modified based on the results of violation handling along scientific workflow execution. Therefore, its initial value has very limited impact. The process of adaptive modification for $PT$ is as follows.

***Definition 9:*** *(Adaptive Modification Process for Probability Threshold).*

Given current probability threshold $PT$ $(0 < PT < 1)$ and checkpoint $a_p$, i.e. a temporal violation is detected at $a_p$, $PT$ is updated as $PT * (1 + \gamma)$. Afterwards, based on our handling point selection rule, if $a_p$ is not selected as a handling point, then $PT$ is updated as $PT * (1 - \gamma)$; otherwise, $PT$ remains unchanged. Here, $\gamma (0 < \gamma < 1)$ stands for the update rate which normally starts from a moderate value such as 0.5 and then gradually decrease to and stay around a small percentile such as 0.05.

The adaptive modification process is to increase the probability threshold $PT$, i.e. the probability of violation handling, where violation handling is triggered; or to decrease $PT$ where violation handling is skipped if self-recovery applies.

In general, our adaptive temporal violation handling point selection strategy is to apply the adaptive modification process for probability threshold to the handling

[1] Note that other distribution models can also be applied based on statistical analysis. However, the strategy for finding the best model is out-of-scope of this paper which can be investigated as a future work.

point selection rule. Table I describes the adaptive violation handling point selection strategy. Our strategy has 5 input parameters, viz. the current checkpoint $a_p$ which is selected by $CSS_{TD}$, the maximum probability time deficit and the minimum probability time redundancy at $a_p$ which are specified according to **Definition 4** and **Definition 6** respectively, the current probability threshold for self-recovery, and the Boolean value for the result of the last temporal violation handling. The output is the decision for whether the current checkpoint will be selected as a handling point or not, i.e. whether a violation handling is necessary or not.

The first step of our strategy is the adaptive modification process for probability threshold. The second step is a comparison between the probability of self-recovery and the probability threshold to determine whether the current checkpoint needs to be selected as a temporal violation handling point.

Table I. Adaptive Temporal Violation Handling Point Selection Strategy

| | |
|---|---|
| **Input** | A necessary and sufficient checkpoint $a_p$ selected by $CSS_{TD}$;<br>The maximum probability time deficit $MPTD(a_p)$;<br>The minimum probability time redundancy $MPTR(a_p)$;<br>The probability threshold for self-recovery $PT$;<br>The result of last temporal violation handling $Success \in (True, False)$. |
| **Output** | True or False as a temporal violation handling point |
| **Step 1** | **Adaptive Modification of** $PT$ |
| ***if (*** $Success == True$ ***)***<br>$PT = PT(1 + \gamma)$ *// if the last violation handling is successful, the current probability threshold is increased according to a predefined speed*<br>***else***<br>$PT = PT(1 - \gamma)$ *// if failed, decreased according to a predefined speed* | |
| **Step 2** | **Temporal Violation Handling Point Selection** |
| $P = F(MPTR(a_p) - MPTD(a_p)) = F(T) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{T} \ell^{\frac{-x^2}{2}}$<br>***if (*** $P > PT$ ***)***<br>**return False //** i.e. $a_p$ ***is not selected as a temporal violation handling point*** *if the current probability of self-recovery is larger than the probability threshold, and temporal violation handling is skipped*<br>***else***<br>**return True //** i.e. $a_p$ ***is selected as a temporal violation handling point*** *and temporal violation handling is required* | |

Our strategy is based on the results of $CSS_{TD}$ where the time overhead is negligible [Chen and Yang 2008b], the computation for the minimum probability time redundancy according to **Definition 6** can be included in the checkpoint selection process [Liu et al. 2011c], *PT* and *Success* are variants. Therefore, the only extra cost for computing the input parameters is for the maximum probability time deficit according to **Definition 4**. After the input parameters are specified, the

current probability of self-recovery is obtained according to **Definition 7** and compared with the adaptively updated probability threshold to further decide whether a handling point will be selected or not. The time overhead for our violation handling point selection strategy can be viewed negligible since it only requires several steps of simple calculation. In the next section, we will evaluate the performance of our strategy comprehensively.

## 5. EVALUATION

In this section, compared with other representative strategies, we demonstrate and validate the results of large scale comprehensive simulation experiments to verify the excellent performance of our adaptive temporal violation handling point selection strategy in reducing violation handling cost while maintaining satisfactory temporal conformance. The full details for the experiments and other related materials can all be found online[2].

### 5.1 Simulation Environment

SwinDeW-C (Swinburne Decentralised Workflow for Cloud) [Liu et al. 2010b] is a prototype cloud workflow system running on SwinCloud, a cloud computing test bed. SwinCloud comprises of many distributed computing nodes. Each node contains many computing units. The primary hosting nodes include the Swinburne SUCCESS (Centre for Computing and Engineering Software Systems) Node, the Swinburne ESR (Enterprise Systems Research laboratory) Node, and the Swinburne Astrophysics Supercomputer Node. To simulate the cloud environment, first, VMWare (http://www.vmware.com/) is installed in SwinCloud nodes so that they can offer unified computing and storage resources. Based on VMWare, several groups of virtual machines are customised with different settings on CPU speed and memory space. Meanwhile, resource scalability is realised by dynamic creation and release of virtual machines. Second, we set up data centres on these nodes which can host different cloud services. In each data centre, Hadoop (http://hadoop.apache.org/) is installed to facilitate the MapReduce computing paradigm and distributed data management. More details about SwinDeW-C and SwinCloud can be found in [Liu et al. 2012].

### 5.2 Simulation Experiments and Results

In the existing works, temporal violation handling point selection is regarded the same as checkpoint selection by nature, hence the state-of-the-art checkpoint selection strategy (with the minimum number of selected checkpoints) $CSS_{TD}$ [Chen and Yang 2008b] is implemented as the benchmark for comparison with ours. In addition, to evaluate the effectiveness of our adaptive strategy (denoted as $AD$), a pure random handling point selection strategy denoted as $RA$ is also implemented. $RA$ selects handling points at necessary and sufficient checkpoints in a pure random fashion. The selection rule of $RA$ is as follows: at a necessary and sufficient checkpoint $a_p$ selected by $CSS_{TD}$, a random value $R$ $(0 < R < 1)$ is first generated. Afterwards, $R$ is compared with the predefined fixed confidence threshold $FT$ $(0 < FT < 1)$, if $R$ is larger than $FT$, then $a_p$ is selected as a handling point. Otherwise, $a_p$ is not selected as a handling point. In $RA$, the fixed confidence

[2] http://www.ict.swin.edu.au/personal/yyang/doc/HandlingPointSelection.rar

threshold $FT$ actually decides how much the portion of checkpoints will be selected as handling points. For example, if $FT$ is set as 0.9, then a total of 1-0.9=0.1, i.e. 10% of the checkpoints will be selected as handling points while the others will be skipped without violation handling. Meanwhile, to compare with the results of workflow execution under the natural condition, the strategy denoted as $NIL$, which records the violation rate without any temporal violation handling, is also implemented.

The experimental settings are described in Table II.

It should be noted that many parameters and models have been used in our experiments to generate testing cases. However, the entire searching space for all the parameters is too large to be fully covered. Therefore, as a common practice [Law and Kelton 2007], in our experiments, we adopt two general rules: 1) for static parameters such as the time compensation rate and success rate for violation handling strategies, we choose representative settings based on either the results obtained in the earlier work or real-world scientific workflow statistics/practice; 2) for dynamic parameters such as scientific workflow sizes and activity durations, we explore a large searching space by predefining a series of candidates or generating randomly on-the-fly.

Table II. Experimental Settings

| | |
|---|---|
| **Scientific Workflows** | Scientific workflow size: 1) large workflows: from 2,000 to 50,000 workflow activities; 2) small workflows: from 200 to 2,000 workflow activities;<br>Activity durations: all activity durations follow the uniform distribution model[3]. The mean duration is randomly selected from 30 to 3000 time units;<br>Temporal constraints: the initial build time probability for deadlines are set as 90% according to [Liu et al. 2011a];<br>Workflow segments: the average length of the workflow segments for subsequent activities is set as 1) 20 for large workflows; 2) 5 for small workflows;<br>Random noises: the duration of one selected activity in each workflow segment is increased by 5%, 15% or 25% of its mean in different rounds. |
| **Temporal Violation Handling** | Temporal violation handling strategy: a pseudo strategy with 50% time compensation rate;<br>Size of subsequent workflow segment: randomly selected as 3 to 5;<br>Success rate: the success rate for violation handling is set as 80%. |
| $CSS_{TD}$ | Default values as defined in [Chen and Yang 2011]. |
| $RA$ | The fixed confidence threshold: $FT$ is set as 0.9, i.e. select 10% from the total checkpoints as adjustment points in a pure random fashion. |
| $AD$ | The update rate: $\gamma$ is initially set as 0.5 and gradually decreased to 0.05. |
| $NIL$ | Without temporal violation handling. |

The process structure is similar to the pulsar searching example presented in Section 2 where both high-level activities and sub-processes (represented by workflow segments which consist of sequential activities) are generated. Here, a workflow activity is an elementary activity such as a basic computation task or a

[3] Note that we have tested our strategy with other distribution models such as normal, exponential and a mixture of them. The results are included in the online documents and the conclusions are consistent.

data transfer task. To evaluate the average performance, 10 independent experiments with different typical scientific workflow sizes (ranging from 2,000 to 50,000 workflow activities for large workflows with their mean activity durations between 30 to 3,000 time units) are executed 100 times each. Note that in **Definition 7** for the probability of self-recovery, normal distribution model is used when no prior knowledge is available. Here, to avoid the potential risk of "self-fulfilling", we use uniform distribution model instead to generate the activity durations. Meanwhile, as included in our online documents, the experimental results for other distribution models such as normal, exponential, and a mixture of them are also available, and the conclusions are consistent. Note that in our previous work [Liu et al. 2011c] for checkpoint selection, both large workflows and small ones such as 200, 500 and 1,000 have been investigated under very similar experimental settings. The results are consistent. Similarly, in this paper, we have also investigated the performance our strategy for small workflows (ranging from 200 to 2,000 workflow activities). Here, the size of 2,000 is included as the borderline for large and small workflows to ensure the continuity of the experiments. The experimental setup for small workflows is exactly the same except that the size of the workflow segment scales down correspondingly from 20 to 5 so as to ensure that the benchmark strategy $CSS_{TD}$ can handle smaller workflow segments and still maintain the close to 0% violation rates [Liu et al. 2011c].

The constraint setting utilises the strategy introduced in [Liu et al. 2011a]. As mentioned in Section 3, this paper only focuses on recoverable temporal violations. Therefore, the initial probability is set reasonably as 90% to serve as a type of QoS contract between the user and the service provider agreed at scientific workflow build time. Note that according to our setting strategy, every workflow activity is covered at least by three types of upper bound constraints: the global constraint, the local milestones, and the individual constraints. Therefore, the temporal dependency exists by nature. Here the initial probability means that a scientific workflow has a 90% probability to finish on time, or in other words, 90% workflow instances can finish on time. Therefore, in our experiment, we specify the "satisfactory temporal conformance" as an acceptable temporal violation rate below 10% so as to meet the QoS contract. As for the actual acceptable temporal violation rate, it can be very different from application to application in the real world. In general, for those applications with hard deadlines (such as the weather forecast scientific workflow mentioned in the introduction), the temporal violation rate needs to be kept very low. However, for those applications with soft deadlines (such as our pulsar searching example and some other scientific workflows which need to process a large size of historic datasets in biology and geology, whether the final results can be delivered exactly on-time would not affect much of their usefulness), the users are much more tolerable with the temporal violation rate but it still needs be within a reasonable range for QoS.

In practice, the initial probability can be a value either smaller or larger than 90%. To make the best use of our strategy, we recommend a practical range for the initial probability which is (90%, 99.87%). Here, its lower bound is set as 90% as a reasonable user requirement or standard service quality for temporal conformance [Liu et al. 2011a], its upper bound is set as 99.87% according to the "$3\sigma$" rule as presented in Section 3. The user may demand the initial probability over 99.87%, but a much higher service price will normally be charged by the cloud service provider since additional resources are likely required to ensure higher QoS. Once outside the practical range, if the initial probability is extremely low (e.g. much lower than 90%),

a much larger number of violation handling is required. In such a case, the cost reduction rate would be very small because most of the checkpoints would also be selected as handling points. Another situation is when the initial probability is extremely high (e.g. very close to 100%), most of the checkpoints can be safely skipped because on-time completion is statistically guaranteed. In such a case, the cost reduction rate would be very large and the violation rate would be very small. However, in such a case, the total number of reduced checkpoints would be very small because there are very few temporal violations. In brief, our strategy can be applied to scientific workflows with different initial probabilities. Even in extreme cases, its performance can still be comparable to that of $CSS_{TD}$, which is the benchmark for our work.

The average length of the workflow segments is set as 20 which is a moderate size for a workflow sub-process similar to those high-level activities depicted in Figure 1. We demonstrated that it had little impact on the number of selected checkpoints in our earlier work [Liu et al. 2011c] where different sizes of workflow segments such as 10, 20, 30, 40 and 50 were tested. Meanwhile, given a specific checkpoint, as presented in Section 4.2, our strategy can determine whether it needs to be selected as a violation handling point only based on the probability of self-recovery and the probability threshold, and these two parameters are both irrelevant to the size of the workflow segment. Therefore, it would not affect the performance of our strategy.

Although under normal circumstances, the experiments can be conducted without noises (i.e. 0% noise), to simulate some worse case scenarios, random noises (i.e. a fixed rate of delays at random activities) are also injected to simulate extra delays accordingly along workflow execution due to potential unpredictable causes such as system overload and resource unavailability [Law and Kelton 2007]. For the comparison purpose, four rounds of simulation experiments with different random noise levels of 0%, 5%, 15% and 25% are conducted respectively to investigate the effectiveness of our strategy from normal to extreme situations.

As for temporal violation handling, there are many violation handling strategies which can be utilised to tackle temporal violations such as workflow local rescheduling, extra resource recruitment and workflow restructure, and their capacity in reducing the occurring time deficits can be different. In general, temporal violation handling is to compensate for the time deficit by reducing the scheduled execution times of the subsequent workflow activities after the handling point. Therefore, in order to be generic and focus on the results of violation handling point selection, instead of applying a specific violation handling strategy in our experiments, a pseudo strategy with a reasonable average time compensation rate (i.e. the reduced rate of the scheduled execution time for the subsequent activities) of 50% is applied to represent the average performance of representative temporal violation handling strategies [Liu et al. 2011c; Yu and Buyya 2007]. The number of activities in the subsequent workflow segment is randomly selected up to 5 which is normally large enough to compensate for time deficit with the given compensation rate. Meanwhile, in the real world, not every workflow local rescheduling can be successful (for example, when the current workload is extremely high). As observed in our previous work [Liu et al. 2011c], around 80% of the time, the local workflow rescheduling strategy is able to successfully compensate the time deficits within 3 to 5 subsequent activities (same as the settings in our experiments). Therefore, 80% is selected as the representative success rate. In real-world systems, the success rate can be either smaller or larger depending on the system environments and the specific violation handling strategies. On one hand, if it is higher than 80%, the

performance of our strategy would be better because time deficits can be statistically compensated with fewer violation handling points. On the other hand, if it is lower than 80%, with the adaptability of our strategy, the number of violation handling points would increase because more violation handling is required. Hence the violation handling cost would also increase accordingly. However, in practice, the strategy with low average success rate (e.g. below 50%) should normally not be used in the first place because in this case, a better strategy should be deployed instead, or alternatively, the users should be suggested to relax the current temporal constraints.

$CSS_{TD}$ is applied with default values as defined in [Chen and Yang 2008b]. The fixed confidence threshold for *RA* is set as 0.9 (i.e. select 10%) so as to be comparable to our strategy. The initial probability threshold and the update rate for our strategy are set as 0.5 (mid-range initially) and 5% (a small percentile) respectively. This is a moderate and reasonable setting when there is no prior knowledge. Additionally, for the comparison purpose, the results of *NIL* (i.e. without any violation handling) are also presented.

#### 5.2.1 Detailed Overview of Experimental Results for Large Workflows

In this section, we demonstrate the four rounds of experiments for large workflows in details. The results for small workflows will be presented separately in Section 5.2.2.

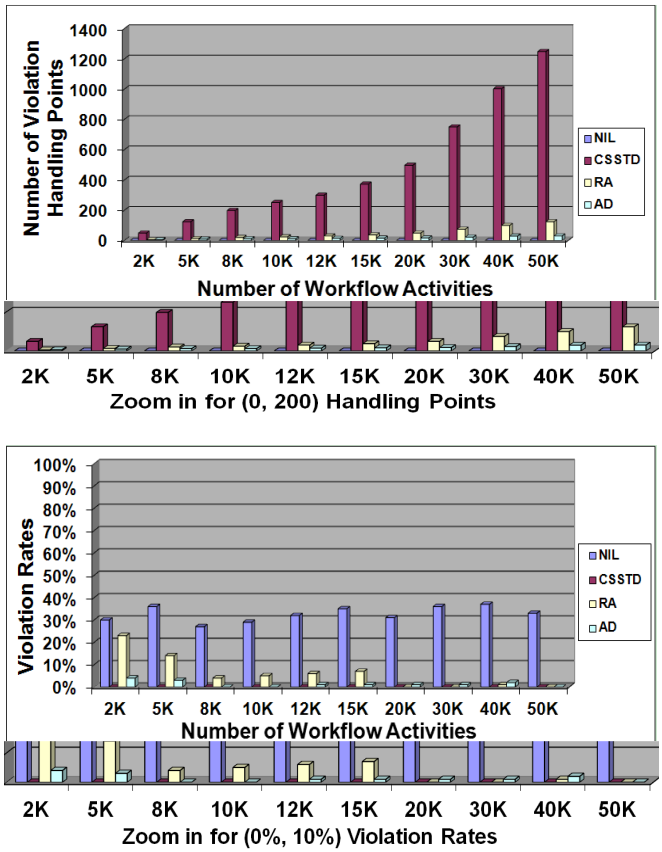
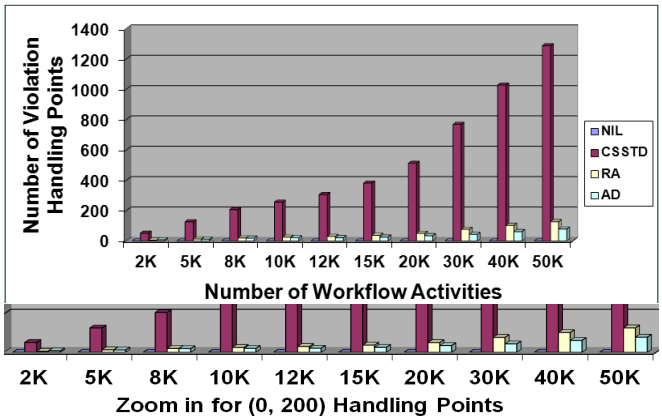
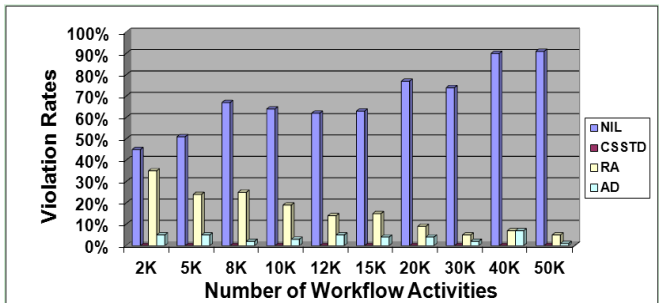



Fig. 5. Experiment results for large workflows (Noise=0%).

The experimental results with 0% noise are depicted in Figure 5. To ensure the clarity of the details, we have zoomed in for the range of (0, 200) handling points and the range of (0%, 10%) violation rates. The case of 0% noise should be the norm in the real world when the system is under a reasonably smooth and stable condition. However, even in such a normal circumstance, the number of checkpoints (i.e. handling points) selected by $CSS_{TD}$ (the time dependency based checkpoint selection strategy, i.e. the benchmark strategy) increases rapidly with the growth of scientific workflow size. For *RA* (the random handling point selection strategy), since the fixed confidence threshold is set as 0.9, it chooses 10% of the handling points selected by $CSS_{TD}$. *NIL* (no selection) does not select any handling points. As for our adaptive handling point selection strategy, *AD*, it selects a very small portion, i.e. 3.5%, of those selected by $CSS_{TD}$. This is a significant reduction, i.e. 96.5% of the violation handling cost over $CSS_{TD}$ in terms of cost effectiveness.

Meanwhile, we compare the violation rate of each strategy. The violation rate of *NIL* is around 32.6%. As for $CSS_{TD}$, the violation rate can be kept close to 0%. The violation rates for *RA* and *AD* are around 23% and 4% respectively for the scientific workflow with the size of 2,000, and they are getting closer to 0% violation rate when the workflow size is becoming larger. This seems to be reasonable at first considering the nature of our strategy is to skip unnecessary violation handling as much as possible, so if the workflow size is larger, there are higher chances that the occurring time deficits can be compensated for at a later stage. However, as will be shown in the following three rounds of experiments, when there are some noises (i.e. the system performance becomes more dynamic), such a trend does not exist anymore.



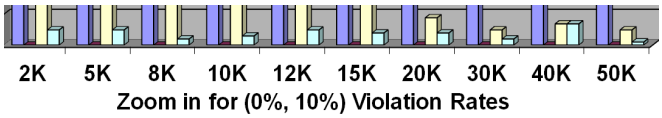



Fig. 6. Experiment results for large workflows (Noise=5%).

The results with 5% random noise are depicted in Figure 6. The number of selected handling points by *AD* is around 6.6% (namely 93.4% cost reduction) of $CSS_{TD}$. In terms of temporal violation, the violation rate for *NIL* is much higher than others which increases to over 90% when the workflow size is larger than 40,000. The violation rate for $CSS_{TD}$ can still be kept close to 0%. As for *RA*, the violation rate is generally higher than the previous round. But when the workflow size grows, the violation rate decreases since violation handling is conducted for many more times. In contrast, our strategy *AD* behaves stably and keeps an average temporal violation rate at only around 3.8%.

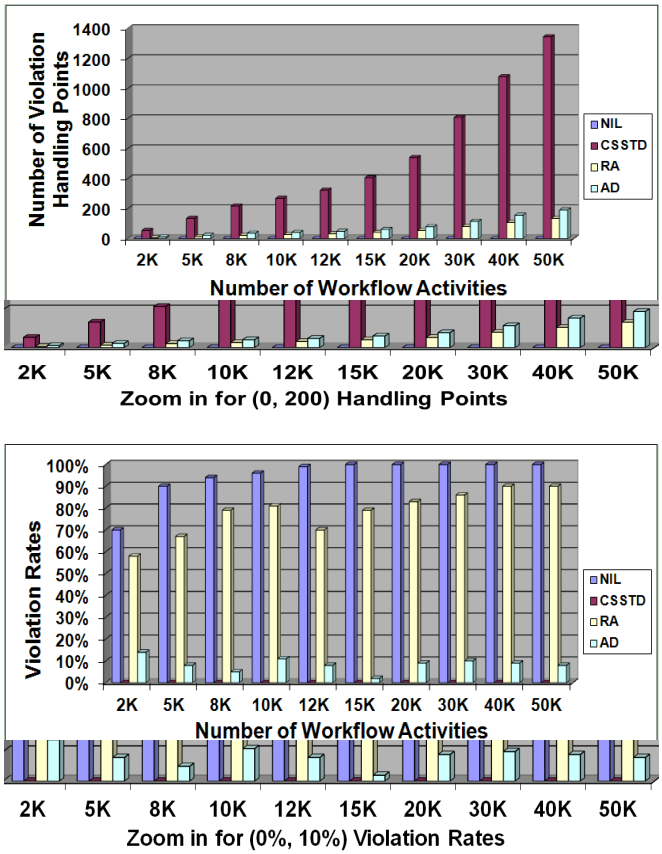



Fig. 7. Experiment results for large workflows (Noise=15%).

Figure 7 shows the results with 15% random noise. *AD* adaptively selects more handling points than that in the previous round due to the increase of the noises. On average, the number of handling points selected by *AD* is 32.3% more than that of *RA*, and it is around 14.7% (namely 85.3% cost reduction) of $CSS_{TD}$. In terms of temporal violation, the violation rate for *NIL* is close to 100% for all the workflows

larger than 15,000. In contrast, $CSS_{TD}$ can still keep the violation rate close to 0%. The violation rate for *RA* is above 78% and increases to over 90% when the workflow size is above 30,000. As for *AD*, since it can adapt to the dynamic system performance, the violation rate is still stable with an average violation rate only increasing to around 8.4%.

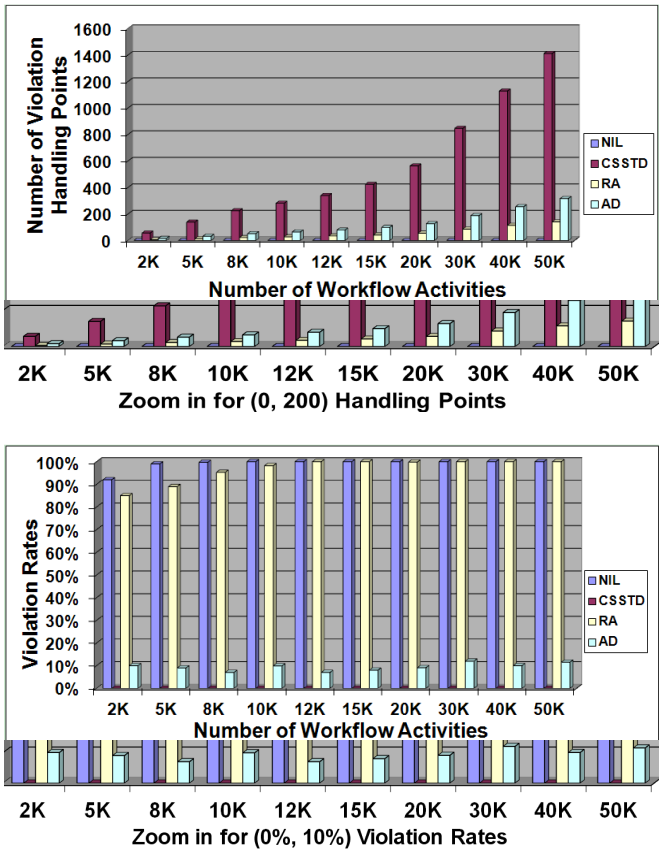



Fig. 8. Experiment results for large workflows (Noise=25%).

Figure 8 shows the results with a rather extreme 25% random noise. The number of selected handling points by *AD* is generally larger, actually with an average of 56% more than that of *RA*, and it is around 22.7% (namely 77.3% cost reduction) of $CSS_{TD}$. In terms of temporal violation, the violation rate for *NIL* is clearly close to 100% for all the cases. $CSS_{TD}$ can still keep the violation rate close to 0% at the cost of a large amount of violation handling. The violation rate for *RA* is now also close to 100% for most workflows larger than 10,000. However, *AD* can still behave stably with an average temporal violation rate of around 9.4% under such an extreme situation, which still meets the "satisfactory temporal conformance" with smaller than 10% violation rate.

As shown in Figure 9, against the benchmark of $CSS_{TD}$, the cost reduction rates of our strategy are around 96.5% for 0% noise; 93.4% for 5% noise; 85.3% for 15% noise and 77.3% for 25% noise respectively; and the corresponding violation rates are around 1.3% for 0% noise; 3.8% for 5% noise; 8.4% for 15% noise and 9.4% for 25%

noise respectively, all below 10% violation rate which denotes satisfactory temporal conformance. The average cost reduction rate is 88.1% and the average violation increase rate is 5.7%. Note that we have also observed the time overruns in every workflow instance with our strategy. The results show that even when those temporal violations take place, the time overruns are often very small in comparison to the overall temporal constraint which can most likely be tolerated by users (see footnote 2 for more details). In overall terms, the results effectively demonstrate that our adaptive temporal violation handling point selection strategy can significantly reduce the violation handling cost while maintaining satisfactory temporal conformance, i.e. lower than the temporal violation rate of 10% which is the agreed QoS contract between users and service providers before workflow execution. In particular, the results for 0% noise with the 96.5% cost reduction rate yet the 1.3% violation rate is very promising in terms of cost effectiveness and temporal conformance because this is the normal circumstance.

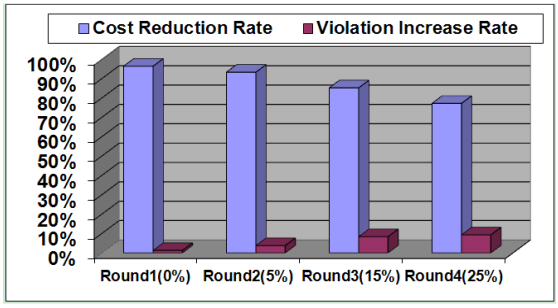



Fig. 9. Cost reduction rate vs. violation increase rate (for large workflows).

### 5.2.2 Brief Overview of Experimental Results for Small Workflows

In this section, given the similarity to the preceding section, we only briefly demonstrate the experimental results for small workflows which are under the same experimental settings except that the size of the workflow segment scales down from 20 to 5 to ensure the performance of the benchmark strategy.

The results for the four rounds of experiments with 0%, 5%, 15% and 25% noises are shown in Figure 10.a, Figure 10.b, Figure 10.c and Figure 10.d respectively. As can be seen from these figures, the variations for the number of the handing points and the violation rates are very similar compared to the results for large workflows. However, it should be noted that due to the decrease of the workflow segment size (i.e. from 20 to 5), checkpoint selection for small workflows is conducted at a much more fine-grained level than large workflows, and hence more checkpoints are selected to ensure the close to 0% violation rates by $CSS_{TD}$. But, the performance of our strategy is stable despite such a change. To demonstrate such an effect, the workflow size of 2,000, as the borderline between large and small workflows, is investigated with the workflow segment size of both 20 and 5. For example, as shown in Figure 5 (round 1 for large workflows), the number of selected checkpoints by $CSS_{TD}$ is 49.3 while its counterpart is 199.2 as shown in Figure 10.a (round 1 for small workflows). However, the corresponding number of violation handling points selected by our $AD$ strategy is 5.9 for the former and 6.1 for the latter, which is very similar. Such a result clearly demonstrate the fact that our violation handling strategy is capable of selecting only *Key* points from the large set of checkpoints to conduct necessary violation handling.

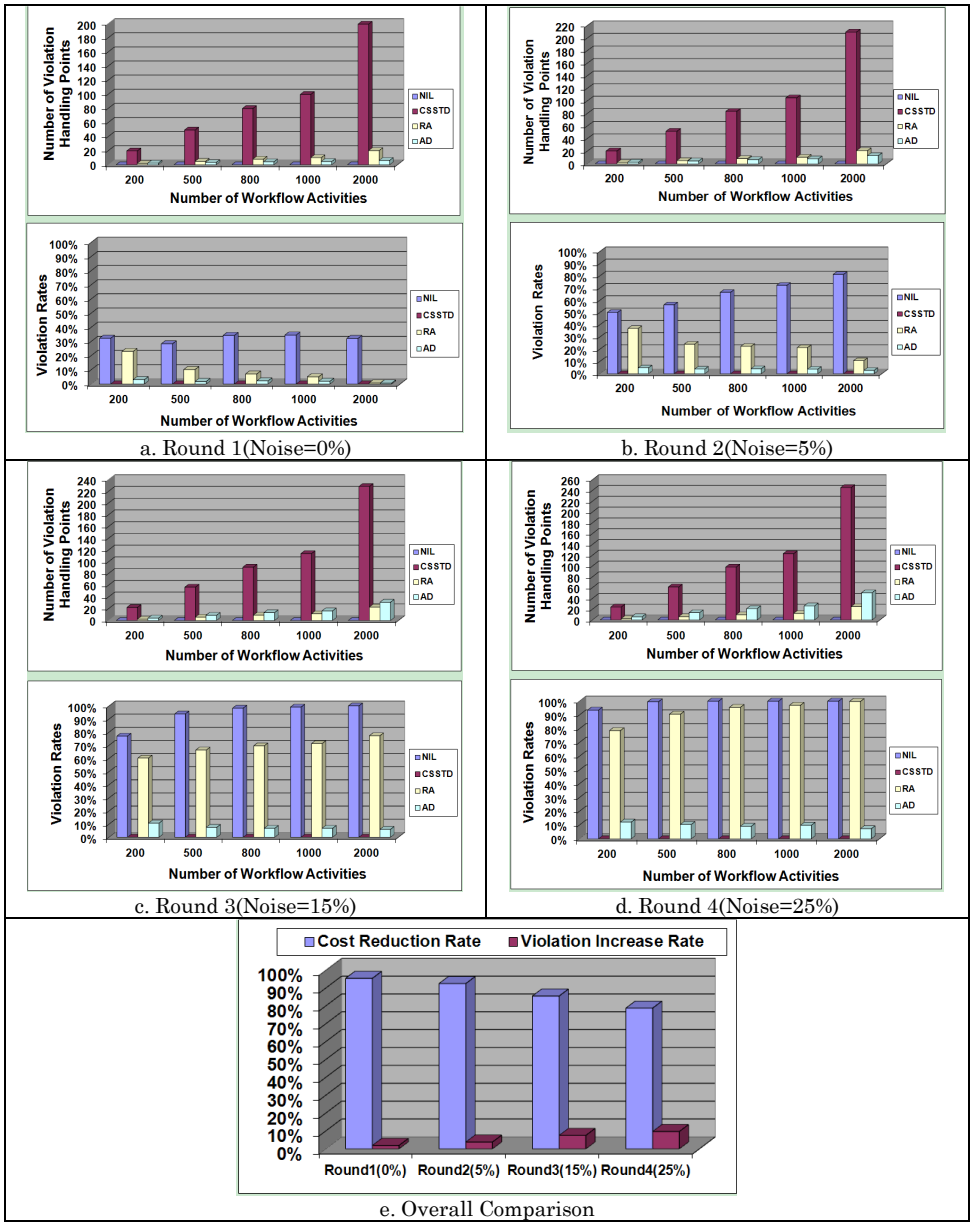



Fig. 10 Experimental results for small workflows.

Specifically, as shown in Figure 10.e, against the benchmark of $CSS_{TD}$ , the cost reduction rates of our strategy for small workflows are around 95.5% for 0% noise; 92.6% for 5% noise; 85.6% for 15% noise and 78.8% for 25% noise respectively; and the corresponding violation rates are around 1.9% for 0% noise; 3.8% for 5% noise; 7.6% for 15% noise and 9.7% for 25% noise respectively, all below 10% violation rate which denotes satisfactory temporal conformance. The average cost reduction rate is 88.1% and the average violation increase rate is 5.8%. It is evident that the results for small workflows are very similar to that of large workflows presented in Section 5.2.1.

Therefore, we can claim that the effectiveness of our strategy is consistent no matter for large or small workflows.

Based on the experimental results demonstrated in the above two sections and those included in our online documents, we can list some of the important facts as follows:

1) Based on the experimental results with different distribution models (e.g. uniform, normal, exponential, and a mixture of them), we can conclude that the effectiveness of our strategy is consistent regardless the actual distribution models.
2) Based on the experimental results with different workflow sizes (from 200 to 50K), we can conclude that the effectiveness of our strategy is consistent regardless the workflow sizes.
3) The best result achieved in our experiments is 98.7% cost reduction rate and close to 0% violation rate (when activity durations follow normal distribution and there is no noise), while the worst result achieved in our experiment is 77.3% cost reduction rate and 9.4% violation rate (when activity durations follow uniform distribution and there are 25% noises).

To summarise, the experimental results demonstrated in this paper show that our strategy can significantly reduce the cost on temporal violation handling by over 96% while maintaining extreme low (about 1.3%) violation rate with marginal overruns under normal circumstances (with uniform distribution and 0% noise). Therefore, with fundamental requirements of *temporal conformance* and *cost effectiveness*, we can conclude that our strategy performs much better than other representative temporal violation handling point selection strategies with different system performance varying from smooth to non-smooth situations.

### 5.3 ANALYSIS ON COST REDUCTION

The above experiments demonstrate the cost reduction rate which denotes the overall percentage of monetary cost and time overhead that can be reduced. In this section, we will further analyse how much monetary cost (in dollars) and time overhead (in CPU hours) can be actually reduced. However, since every scientific workflow system may reside in a specific system environment with different resource capabilities, pricing models (e.g. the price for computing, storage and network devices), and the choices of temporal violation handling strategies, it is not practical, and unnecessary, for us to conduct cost analysis for every situation. Therefore, we illustrate with representative experimental settings.

Table III. Settings for cost analysis

| **Workflow Settings** | |
|---|---|
| Data Size | For each workflow activity, its data size (the input data required for their execution) are generated within a range of (0.1GB~0.5GB) |
| Other Settings | Same as described in Table II |
| **Resources and Prices** | |
| Computation Speed | The computation speed for each resource is defined as 1, i.e. all the workflow activities are completed with their average duration as defined in Table II. |
| Computation Price | The computation price is $2.00 per hour (High-Memory Quadruple Extra Large on-demand instances). |
| Network Speed | The network speed is 1GB/s (Standard Gigabyte Ethernet in cloud data centres). |
| Network Price | The network price is $0.05 per GB. |

In general, for the running of scientific workflows, the cost is mainly generated by the usage of three types of resources, viz, computation, storage and network. Here, since we only consider the extra cost brought by violation handling, the cost for data storage could not change, hence not be included. In contrast, the costs for computation and network could be included since they require extra usage of resources. Therefore, in our experiments, we focus on the costs for computation and network only.

In order to estimate the violation handling cost, additional settings on the workflow activities and resources have been presented in Table III. The data size for each workflow activity is randomly generated within the range of 0.1GB to 0.5GB which is the primary range for intermediate date sets in the pulsar searching example. The other settings on scientific workflows are the same as shown in Table II. The resource prices are set based on the Amazon Web Services (AWS) pricing model (http://aws.amazon.com/pricing/) where AWS is one of the world's largest commercial cloud computing service providers. In the AWS cloud, computation activities are fulfilled by the EC2 services (Elastic Compute Cloud, http://aws.amazon.com/ec2/) which can provide different levels of compute capacities. For example, in the pulsar searching workflow, to execute data and computation intensive activities, we would need to choose the High-Memory Quadruple Extra Large Instance which has 68.4 GB of memory, 26 EC2 Compute Units (8 virtual cores with 3.25 EC2 Compute Units each), and 1690 GB of local instance storage. This type of EC2 service is comparable to the compute capacity of a computing node with 2 quad-core processors in our Swinburne Supercomputer node as mentioned in Section 5.1. The price for EC2 services may vary in different regions. For a High-Memory Quadruple Extra Large Instance in US East (Virginia), the price is $2.00 per hour. For data transfer, AWS offers several options with different speeds and prices such as Internet transfer, internal network transfer and the novel AWS Import/Export (http://aws.amazon.com/importexport/) service which can use portable storage devices and transfer the data onto and off of storage devices. For conventional network transfer, AWS has a complex pricing model where the prices are decreasing with the increase of the data volume. Therefore, according to the data size for our pulsar searching example, we choose the following setting for the network resources which is consistent to the AWS pricing model. Specifically, the network speed is set as 1GB/s which is a comparable speed for a standard Gigabyte Ethernet in the cloud data centres, and the corresponding network price is $0.05 per GB.

Note that in some computing environments such as scientific community grid, computing and network resources are used by community members for free, normally after allocation of resource consumption quotes. In such cases, there will be no explicit monetary cost on the user's side as it is on system's side. Meanwhile, the time overhead is critical in any computing environment since it contributes to the overall completion time of scientific workflows.

Based on the settings in Table III, we demonstrate two basic scenarios respectively with two representative violation handling strategies, viz. workflow local rescheduling, and extra resource recruitment. At the moment, we only demonstrate the results for a single violation handling. Given there could be a large number of violations, the overall cost reduction can be estimated with the total number of violation handling points.

**Scenario 1**: Workflow local rescheduling

Workflow local rescheduling is to compensate for the time deficit by optimising the current activity-resource assignment. The major monetary cost and time

overhead involved in workflow local rescheduling are generated by activity transfer (including input data, activity definitions, and programs if required) between resources, and the computation of the rescheduling strategy. The workflow local rescheduling strategy adopted here is an ACO based two stage workflow local rescheduling strategy (ACOWR) which optimises the integrated task-resource list including both the activities of the violated workflow segment and their co-located activities [Liu et al. 2010a]. As shown in Table IV, to accommodate the requirements of ACOWR, three test cases: Case 1, Case 2 and Case 3 with increasing numbers of workflow activities, local workflow segments (i.e. a small set of 3 to 5 activities with precedence relationship) and workflow resources are generated.

Table IV. Cost for workflow local rescheduling

| Items | Case 1 | Case 2 | Case 3 |
|---|---|---|---|
| Number of activities | 100 | 180 | 300 |
| Number of segments | 12 | 25 | 50 |
| Number of resources | 6 | 10 | 20 |
| Computation cost | $\$1.34*10^{-3}$ | $\$2.6*10^{-3}$ | $\$5.3*10^{-3}$ |
| Transfer cost | \$0.318 | \$0.564 | \$0.942 |
| **Total cost** | \$0.319 | \$0.567 | \$0.947 |
| Computation time | 2.42s | 4.75s | 9.54s |
| Transfer time | 3.17s | 5.65s | 9.42s |
| **Total time** | 5.59s | 10.40s | 18.96s |
| **Average cost** | **\$0.60** | | |
| **Average time** | **12.0s** | | |

Meanwhile, during the execution of scientific workflows, intermediate data can be handled in two basic ways. In the first way, after the completion of the predecessor activities, the produced intermediate data will be transferred according to the routing of the scientific workflow, i.e. the data will be passed to the applications in charge of processing the successor activities. In the second way, the intermediate data will be stored in the same location over the entire workflow lifecycle. All the successor activities will read the intermediate data, process it, and then write back the execution results. In scientific workflows, if the data needs to be processed locally by the application, the intermediate data will normally be handled in the first way. Otherwise, the intermediate data will normally be handled in the second way. Accordingly, in the first way, after workflow local rescheduling, the intermediate data needs to be moved together with its corresponding activity when it has been re-allocated, which may incur extra cost on data transfer. In contrast, in the second way, there is no extra cost since the intermediate data will be read from the original location before and after workflow local rescheduling. In our experiments, considering the fact that the second way dominates most of the scientific workflows, and based on the Pareto principle (also known as the 80-20 rule) [Law and Kelton 2007], 80% of the intermediate datasets are specified as handled in the second way while the other 20% in the first way.

The total monetary cost and total time overhead required for a single workflow local rescheduling are listed in Table IV. Here, the total monetary cost for a single workflow local rescheduling is equal to the sum of the computation cost (i.e. the cost for running the rescheduling algorithms) and the transfer cost (i.e. the cost for necessary data transfer); the total time overhead for a single workflow local rescheduling consists of the computation time (i.e. the time for running the rescheduling algorithms) and the transfer time (i.e. the time for data transfer). The

results show that for workflow local rescheduling, the major cost component is the cost generated by data transfer, while the computation cost for the rescheduling algorithms is very minor in comparison. This is consistent with the current experience in running applications in the cloud where the bottleneck of the system performance is usually the network. Based on such results, we can have the conclusion that for a typical single workflow local rescheduling scenario (with around 200 activities and 10 resources), the average monetary cost is around $0.60 and the average time overhead is around 12 seconds.

**Scenario 2**: Extra resource recruitment

As for extra resource recruitment, it is to compensate for the time deficit by employing additional resources for the violated workflow instances at runtime. Its major monetary cost and time overhead are generated by the set-up of new resources (recruit and deploy) and the data transfer. For extra resource recruitment, adding a new resource is the most common way to handle temporal violations. In our experiments, similar to the local workflow segment rescheduled by ACOWR in Scenario 1, an average size of 3~5 activities are transferred to a new resource.

The total monetary cost and total time overhead required for adding a new resource are listed in Table V. The total monetary cost for adding a new resource consists of three components, viz. the set-up cost, the computation cost and the transfer cost. Since the AWS cloud adopts the "pay-as-you-go" model, it is free for setting up new resources. Meanwhile, since the computation cost would not change for executing the re-allocated activities, it will not generate extra charge (denoted as Nil in the table). Therefore, the only extra charge is the transfer cost. The total time overhead consists of the set-up time for a new service and the transfer time for the data. In the AWS cloud, it is usually around 5 minutes to request a new EC2 service instance and deploy necessary software services. Therefore, in our experiments, we assign the set-up time as 5 minutes, i.e. 300 seconds. Clearly, the set-up time is much bigger compared with the data transfer time in our three cases. This is consistent to the current situation where adding new resources at runtime (named as "on-demand instance") produces large time overhead. To overcome such a problem, EC2 also offers another choice named the "reserved instance" (http://aws.amazon.com/ec2/reserved-instances/) which is standby for immediate use. However, reserved instances are charged in advance for a fixed term such as 1-year or 3-years. Therefore, it can become a type of waste if the reserved resources are not well utilised. Clearly, this is a trade-off between cost and time. In this paper, we only consider the on-demand instance for illustration. Based on such results, we can have the conclusion that for a typical single workflow local rescheduling scenario, by adding a new resource, the average total cost is around $0.09 and the average total time is around 301.0 seconds.

Table V. Cost for adding a new resource

| **Items** | **Case 1** | **Case 2** | **Case 3** |
|---|---|---|---|
| Number of activities | 3 | 4 | 5 |
| Set-up cost | Free | Free | Free |
| Computation cost | Nil | Nil | Nil |
| Transfer cost | $0.056 | $0.076 | $0.142 |
| **Total cost** | $0.056 | $0.076 | $0.142 |
| Set-up time | 300s | 300s | 300s |
| Transfer time | 0.57s | 0.77s | 1.42s |
| **Total time** | 300.6s | 300.8s | 301.4s |
| **Average cost** | **$0.09** | | |
| **Average time** | **301.0s** | | |

Based on the above results, we can see that in the AWS cloud, workflow local rescheduling is relatively expensive but has very small time overhead, while adding a new resource is much cheaper but has a very large time overhead. Therefore, as for which strategy should be applied, it is an interesting trade-off that needs to be decided. However, this is beyond the scope of this paper and hence will be addressed elsewhere in the future. Note that we have also analysed three other representative violation handling strategies including stop and restart, processor swapping and workflow restructure, and the cost generated by them are even larger. Discussions are included in a technical report which can be found online (see footnote 2 on page 17) and hence omitted here. Now we demonstrate how much cost and time may be actually reduced in our motivating example by our strategy based on the results of violation handling point selection and cost analysis of violation handling strategies presented above. It should be noted that other workflows with larger sizes will have higher cost reduction and time reduction because their number of reduced violation handling points is larger as demonstrated in Section 5.2.

The pulsar searching workflow contains over 200 high-level activities (including the 13 parallel paths for different beams) where each may have more than tens of sub-level activities which are elementary activities. Therefore, the pulsar searching workflow can contain more than 2,000 activities. Meanwhile, as introduced in Section 2.1, since each pulsar searching workflow needs to be finished within 24 hours, we assume it runs for a maximum of once a day, i.e. 365 times per year. Based on these figures, we choose the results for the workflow size of 2,000 activities (as long workflows), and calculate how much cost and time can be reduced every year by the implementation of our strategy in the two representative scenarios, viz. workflow local rescheduling and extra resource recruitment.

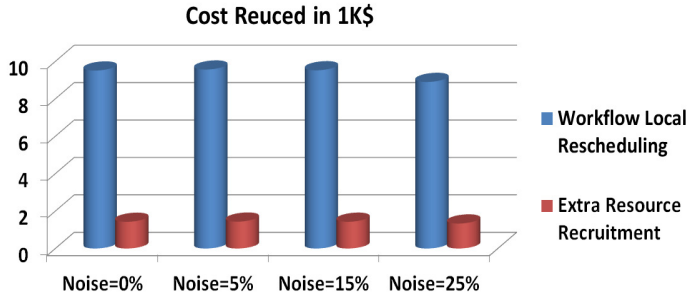



Fig. 11. Yearly cost reduction for the pulsar searching workflow.

Figure 11 depicts the yearly cost reduction for the pulsar searching workflow. As analysed above, for a typical workflow local rescheduling scenario, the average cost is around \$0.6, and for a typical extra resource recruitment scenario, the average cost is around \$0.09. Therefore, since the total number of temporal violations is around 19,000 per year (the average number of temporal violation handling points selected by $CSS_{TD}$ multiplied by 365 runs per year), the yearly cost for temporal violation handling is tremendous. With our adaptive violation handling point selection strategy, the reduced number of violation handling points is around 15,600 per year. Therefore, the yearly cost reduction for violation handling is very impressive. As shown in Figure 11, the reduced cost for each round of experiments is very close to each other. Specifically, the average yearly reduced cost for workflow local rescheduling is around \$9,400, and the average yearly reduced cost for extra resource recruitment is around \$1,400.

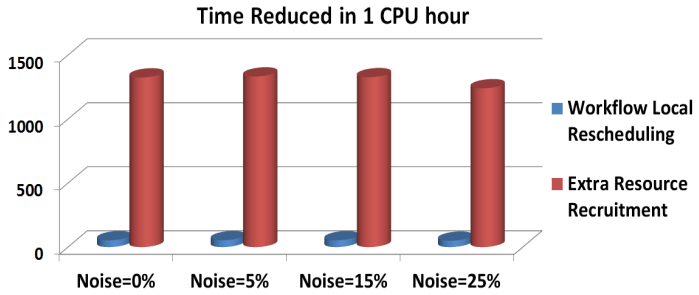



Fig. 12. Yearly time reduction for the pulsar searching workflow.

Figure 12 depicts the yearly time reduction for the pulsar searching workflow. As analysed above, for a typical workflow local rescheduling scenario, the average time is around 12 seconds, and for a typical extra resource recruitment scenario, the average total time is around 301 seconds. Therefore, the yearly time reduction for workflow local rescheduling is relatively smaller. As shown in Figure 12, the reduced CPU hours for each round of experiments are very close to each other. Specifically, the average yearly reduced time for workflow local rescheduling is around 52 CPU hours, and the average yearly reduced time for extra resource recruitment is around 1,306 CPU hours.

To conclude, in scientific workflow systems, no matter with workflow local rescheduling or extra resource recruitment at workflow runtime, the total monetary cost and time overhead for handling temporal violations are non-trivial. Therefore, temporal violation handling should only be conducted unless truly necessary. Our adaptive temporal violation handing point selection strategy plays a significant role in making such a decision and it helps to achieve significant reduction of the overall temporal violation handling cost and time in scientific workflow systems.

### 5.4 Threats to Validity

Here we discuss the threats to validity in our work. We discuss the external threats followed by the internal threats.

*External threats to validity*. The main threat to the external validity of our evaluation is the representativeness of our motivating example. The pulsar searching process is a typical scientific workflow which has the special features such as computation, data and transaction intensity, less human interaction, and a large number of activities [Taylor, et al. 2007]. Therefore, the data collected from the pulsar searching process, especially the statistics for activity durations (viz. the sample means and the sample standard deviations), are representative. But still, for those scientific workflows from other areas, the statistics can be very different, especially the mean durations. But as we can see from the definitions of *Probability Time Deficit* and *Probability Time Redundancy* in Section 4.2, our adaptive violation handling point selection strategy mainly concerns with standard deviations which represent the dynamics of the system environments. The basic idea of our strategy is to fully utilise the time redundancy of each activity (defined by the standard deviation and the probability percentile) to compensate for the time deficit. Therefore, the main external threat, i.e. whether our work can be generalised to other scientific workflows, is the representativeness of the measured standard deviations. We minimised this issue by comparing with one of our earlier work in [Liu et al. 2011a] on a weather forecast scientific workflow. The system performance is very dynamic and very similar to our pulsar searching process. Similar findings are also reported

in other literatures [Prodan and Fahringer 2008]. Therefore, the system dynamics of our simulation experiments should be representative.

*Internal threats to validity*. The main threat to the internal validity of our evaluation is the comprehensiveness of our experiments. Many parameters and models have been used to generate the testing cases and evaluate the performance of our strategy. Given that the entire searching space for all the parameters are too large to be fully covered, therefore, as mentioned at the beginning of Section 5.2, we minimise this issue by adopting two general rules as a common practice [Law and Kelton 2007] in our experimental settings, namely: 1) for static parameters, such as the compensation rate and the success rate for violation handling, we choose representative settings based on either the results obtained in the earlier work or real-world scientific workflow statistics/practice; 2) for dynamic parameters, such as the workflow sizes and activity durations, we explore a large searching space by predefining a series of candidates or generating randomly on-the-fly. Meanwhile, also recommend the practical ranges of parameter settings to make the best use of our strategy. Therefore, our comprehensive experiments have explored a representative and large enough searching space for the parameters.

## 6. RELATED WORK

Time, as one of the most basic measurements for system performance, has long been used and studied in software engineering practice and research [Chinn and Madey 2000; Law and Kelton 2007; Schwalb and Vila 1998]. In the real world, since many workflow processes are required to be completed in a constrained period of time, temporal constraints are among the most important workflow QoS constraints besides cost, fidelity, reliability and security as discussed in [Yu and Buyya 2005]. There are many studies contributing to the management of temporal constraints in workflow systems such as process modelling, verification and validation, monitoring, exception handling and performance analysis [van der Aalst and Hee 2002; van der Aalst et al. 2000; Chen and Yang 2008a; Lerner et al. 2010; Russell et al. 2006b]. The violations of temporal constraints, i.e. failures of in-time completion, can be regarded as a type of non-functional exception. In contrast to the studies on the traditional functional exceptions of workflow systems [Buhr and Mok 2000; Hagen and Alonso 2000], the research on non-functional exceptions, i.e. QoS violations, has attracted increasing interests due to the popularity and the dynamic service quality of distributed computing infrastructures such as grid, p2p and cloud [Ferretti et al. 2010; Krauter et al. 2002; Liu et al. 2010b].

In the area of temporal QoS management in scientific workflow systems, many efforts have been dedicated to different tasks involved in temporal verification. The work in [Liu et al. 2011a] proposes a probabilistic strategy for setting both workflow-level (coarse-grained) and activity-level (fine-grained) temporal constraints based on a negotiation process between users and service providers. As mentioned earlier, a runtime checkpoint selection strategy aims at selecting activity points on the fly to conduct temporal verification and/or violation handling so as to improve the efficiency of monitoring large scale scientific workflows and reduce the computation cost [Chen and Yang 2007a]. Meanwhile, to reduce the violation handling cost, multiple-state based temporal verification is proposed to detect multiple fine-grained temporal violations so that different temporal violations (with different levels of time deficits) can be tackled by different violation handling strategies [Chen and Yang 2007a; Liu et al. 2011c]. Many scheduling and rescheduling algorithms have been investigated and applied in both workflow and general software application processes

to address QoS requirements and exceptions [Yu and Buyya 2007; Yu and Shi 2007]. The work in [Xiao et al. 2010] proposes a multi-objective rescheduling method to address the need for software project resource management in handling different types of exceptions.

In recent years, many checkpoint selection strategies, from intuitive rule based to sophisticated model based, have been proposed. The work in [Eder, et al. 1999] takes every workflow activity as a checkpoint. The work in [Marjanovic and Orlowska 1999] selects the start activity as a checkpoint and adds a new checkpoint after each decision activity is executed. It also mentions a type of static activity point which is defined by users at the build-time stage. There are also some other strategies such as the one which selects an activity as a checkpoint if its execution time exceeds the maximum duration and the one which selects an activity as a checkpoint if its execution time exceeds the mean duration [Chen and Yang 2011]. The checkpoint selection which satisfies the property of necessity and sufficiency is proposed in [Chen and Yang 2007a] where an activity point is selected as a checkpoint if and only if its execution time is larger than the sum of its mean duration and its minimum time redundancy. Based on that, the work in [Chen and Yang 2008b; Chen and Yang 2011] further improves the efficiency of temporal verification by utilising the temporal dependency between temporal constraints. While these previous works are mainly based on the (discrete) multi-state based temporal consistency model, the latest work in [Liu et al. 2011c] utilises the (continuous) probability based temporal consistency model which can take advantages of more complex statistical models and thus to further improve the effectiveness of the checkpoint selection strategy. So far, to the best of our knowledge, the temporal consistency based checkpoint selection strategy is the most efficient checkpoint selection strategy that can be found in the literature. Therefore, it is facilitated as the foundation for our work reported in this paper.

The phenomenon of self-recovery, especially in scientific workflows where the performance of underlying infrastructure is very dynamic, has been overlooked by most researchers. The work in [Chen and Yang 2007b] proposes a light-weight temporal violation handling strategy named TDA (time deficit allocation). Although self-recovery has not been formally defined there, the idea of TDA which utilises the time redundancy of the subsequent activities to compensate for the current time deficit can be regarded similar to self-recovery. However, it is used as a violation handling strategy or as a pre-step for workflow local rescheduling as detailed in [Liu et al. 2011b]. The issue of temporal violation handling point selection has so far not been explicitly proposed in any literature because conventional studies believe that similar to the handing of functional exceptions, every temporal violation in scientific workflow systems needs to be handled. Hence temporal violation handling point selection has not been regarded as a separate task from necessary and sufficient checkpoint selection. However, the non-functional nature of temporal violations has been neglected. Therefore, the research in temporal violation handling point selection is still in its infancy that requires much more efforts along the line.

## 7. CONCLUSIONS AND FUTURE WORK

In the context of temporal violation handling in scientific workflow systems, to reduce the high and rapidly increasing cost for resolving temporal violations, effective temporal violation handling point selection is required to decide whether violation handling should be conducted immediately once a temporal violation is detected. The existing work on workflow temporal management believes that, similar

to functional exceptions, temporal violations should be handled whenever they are detected. Therefore, temporal checkpoints are treated the same as temporal violation handling points, but the non-functional nature of temporal violations has been neglected. However, differing from the state-of-the-art necessary and sufficient checkpoint selection, this paper presents a trade-off between *temporal conformance* and *cost effectiveness* which could be made in temporal violation handling point selection to reduce the high violation handling cost. Therefore, motivated by the phenomenon of self-recovery, i.e. the phenomenon that the occurring time deficit could be automatically compensated for by the time redundancy of the subsequent workflow activities, a novel adaptive temporal violation handling point selection strategy has been proposed based on a probability based runtime temporal consistency model. The results of large scale simulation experiments have demonstrated the cost effectiveness of our strategy as it can significantly reduce the violation handling cost over other representative strategies while maintaining satisfactory temporal conformance. Specifically, our strategy can significantly reduce the cost on temporal violation handling over 96% while maintaining extremely low violation rate with marginal overruns under normal circumstances. Moreover, for our pulsar searching motivating example, the yearly cost reduction can reach an average of $9,400 (with workflow local rescheduling as the selected violation handling strategy), and the yearly time reduction can reach an average of 1,306 CPU hours (with extra resource recruitment as the selected violation handling strategy). Therefore, based on the results demonstrated in this paper, we can change the conventional answer to the question "*do we need to handle every temporal violation in scientific workflow systems*" from "*true*" to "*not necessarily true*". To the best of our knowledge, this is the first work that has proposed and systematically investigated the problem of temporal violation handling point selection in scientific workflow systems.

In the future, we plan to investigate the temporal violation handling cost in other software applications to further illustrate the benefit of the work. We also aim to investigate and improve our strategy for better temporal conformance in the system environments where extremely large noises exist but some prior knowledge may be available. This requires joint efforts of temporal knowledge discovery, temporal violation handling point selection as well as temporal violation handling strategies.

## Acknowledgement

The authors are grateful for some initial discussions on adaptive random testing with Dr. R. Merkel and Dr. H. Liu from Swinburne University Centre for Computing and Engineering Software Systems, on pulsar searching workflow with Dr. W. van Straten and Ms. L. Levin from Swinburne Centre for Astrophysics and Supercomputing, and on Amazon Web Services and its pricing model with Mr. S. Brunozzi and Mr. M. Brown from Amazon Web Services. The authors are also grateful for anonymous reviewers of this paper for improvement.

## References

ALLEN, J.F. 1983. Maintaining Knowledge about Temporal Intervals. *Communication of the ACM, 26*, 832-843.

BUHR, P.A. AND MOK, W.Y.R. 2000. Advanced Exception Handling Mechanisms. *IEEE Transactions on Software Engineering, 26*, 820-836.

BUYYA, R., YEO, C.S., VENUGOPAL, S., BROBERG, J. AND 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*, 599-616.

CHEN, J. AND YANG, Y. 2007a. Adaptive Selection of Necessary and Sufficient Checkpoints for Dynamic Verification of Temporal Constraints in Grid Workflow Systems. *ACM Transactions on Autonomous and Adaptive Systems, 2, article 6*.

CHEN, J. AND YANG, Y. 2007b. Multiple States based Temporal Consistency for Dynamic Verification of Fixed-Time Constraints in Grid Workflow Systems. *Concurrency and Computation: Practice and Experience, 19*, 965-982.

CHEN, J. AND YANG, Y. 2008a. A Taxonomy of Grid Workflow Verification and Validation. *Concurrency and Computation: Practice and Experience, 20*, 347-360.

CHEN, J. AND YANG, Y. 2008b. Temporal Dependency based Checkpoint Selection for Dynamic Verification of Fixed-Time Constraints in Grid Workflow Systems. *Proc. 30th International Conference on Software Engineering*, Leipzig, Germany, 141-150.

CHEN, J. AND YANG, Y. 2011. Temporal Dependency based Checkpoint Selection for Dynamic Verification of Temporal Constraints in Scientific Workflow Systems. *ACM Transactions on Software Engineering and Methodology 20(3), article 9*.

CHEN, T.Y. AND MERKEL, R. 2008. An Upper Bound on Software Testing Effectiveness. *ACM Transactions on Software Engineering Methodology, 17*, 1-27.

CHINN, S.J. AND MADEY, G.R. 2000. Temporal Representation and Reasoning for Workflow in Engineering Design Change Review. *IEEE Transactions on Engineering Management, 47*, 485-492.

DEELMAN, E., GANNON, D., SHIELDS, M. AND TAYLOR, I. 2008. Workflows and e-Science: An Overview of Workflow System Features and Capabilities. *Future Generation Computer Systems, 25*, 528-540.

EDER, J., PANAGOS, E. AND RABINOVICH, M. 1999. Time Constraints in Workflow Systems. *Proc. 11th International Conference on Advanced Information Systems Engineering,* Heidelberg, Germany, 286-300.

FERRETTI, S., GHINI, V., PANZIERI, F., PELLEGRINI, M. AND TURRINI, E. 2010. QoS-Aware Clouds. *Proc. IEEE 3rd International Conference on Cloud Computing*, Miami, Florida, USA, 321-328.

HAGEN, C. AND ALONSO, G. 2000. Exception Handling in Workflow Management Systems. *IEEE Transactions on Software Engineering, 26*, 943-958.

KRAUTER, K., BUYYA, R. AND MAHESWARAN, M. 2002. A Taxonomy and Survey of Grid Resource Management Systems for Distributed Computing. *Software Practice and Experience, 32*, 135-164.

LAW, A.M. AND KELTON, W.D. 2007. *Simulation Modelling and Analysis (Fourth Edition)*. McGraw-Hill.

LERNER, B.S., CHRISTOV, S., OSTERWEIL, L.J., BENDRAOU, R., KANNENGIESSER, U. AND WISE, A. 2010. Exception Handling Patterns for Process Modeling, *IEEE Transactions on Software Engineering, 36*, 162-183.

LIU, X., CHEN, J. AND YANG, Y. 2008. A Probabilistic Strategy for Setting Temporal Constraints in Scientific Workflows. *Proc. 6th International Conference on Business Process Management*, Milan, Italy, 180-195.

LIU, X., CHEN, J., WU, Z., NI, Z., YUAN, D. AND YANG, Y. 2010a. Handling Recoverable Temporal Violations in Scientific Workflow Systems: A Workflow Rescheduling Based Strategy. *Proc. 10th International Symposium on Cluster, Cloud and Grid Computing*, Melbourne, Australia, 534-537.

LIU, X., YUAN, D., ZHANG, G., CHEN, J. AND YANG, Y. 2010b. SwinDeW-C: A Peer-to-Peer Based Cloud Workflow System. In *Handbook of Cloud Computing*, B. FURHT AND A. ESCALANTE Eds. Springer.

LIU, X., NI, Z., CHEN, J. AND YANG, Y. 2011a. A Probabilistic Strategy for Temporal Constraint Management in Scientific Workflow Systems. *Concurrency and Computation: Practice and Experience,23, 1893-1919*.

LIU, X., NI, Z., WU, Z., YUAN, D., CHEN, J. AND YANG, Y. 2011b. A Novel General Framework for Automatic and Cost-Effective Handling of Recoverable Temporal Violations in Scientific Workflow Systems. *Journal of Systems and Software, 84*, 492-509.

LIU, X., YANG, Y., JIANG, Y. AND CHEN, J. 2011c. Preventing Temporal Violations in Scientific Workflows: Where and How. *IEEE Transactions on Software Engineering, ,* 37, 805-825.

LIU, X., YUAN, D., ZHANG, G., LI, W., CAO, D., HE, Q., CHEN, J. AND YANG, Y. 2012. The Design of Cloud Workflow Systems. Springer.

MARJANOVIC, O. AND ORLOWSKA, M.E. 1999. On Modelling and Verification of Temporal Constraints in Production Workflows. Knowledge and Information Systems 1, 157-192.

PDSEC 2009. In The 10th IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing, in conjunction with The 23rd Parallel and Distributed Processing Symposium (IPDPS-09), May 25-29, 2009, Rome, Italy

PRODAN, R. AND FAHRINGER, T. 2008. Overhead Analysis of Scientific Workflows in Grid Environments. IEEE Trans. on Parallel and Distributed Systems 19, 378-393.

RUSSELL, N., AALST, W.M.P.V.D. AND HOFSTEDE, A.H.M.T. 2006a. Exception Handling Patterns in Process-Aware Information Systems BPMcenter.org.

RUSSELL, N., AALST, W.M.P.V.D. AND HOFSTEDE, A.H.M.T. 2006b. Workflow Exception Patterns. In 18th International Conference on Advanced Information Systems Engineering (CAiSE'06) Springer-Verlag, Berlin, Gemany, 288-302.

SCHWALB, E. AND VILA, L. 1998. Temporal Constraints: A Survey. Constraints 3, 129-149.

SECES 2008. In First International Workshop on Software Engineering for Computational Science and Engineering, in conjuction with the 30th International Conference on Software Engineering (ICSE2008), Leipzig, Germany, May, 2008.

SOMMERVILLE, I. 2009. Software Engineering (9th Edition). PEARSON.

STROUD, K.A. 2007. Engineering Mathematics (Sixth Edition). Palgrave Macmillan, New York.

TAYLOR, I.J., DEELMAN, E., GANNON, D.B. AND SHIELDS, M. 2007. Workflows for e-Science: Scientific Workflows for Grids. Springer.

WANG, L., JIE, W. AND CHEN, J. 2009. Grid Computing: Infrastructure, Service, and Applications CRC Press, Talyor & Francis Group.

XIAO, J., OSTERWEIL, L., WANG, Q. AND LI, M. 2010. Dynamic Resource Scheduling in Disruption-Prone Software Development Environments, 107-122.

YANG, Y., LIU, K., CHEN, J., LIGNIER, J. AND JIN, H. 2007. Peer-to-Peer Based Grid Workflow Runtime Environment of SwinDeW-G. In 3rd International Conference on e-Science and Grid Computing (e-Science07), Bangalore, India, 51-58.

YU, J. AND BUYYA, R. 2005. A Taxonomy of Workflow Management Systems for Grid Computing. Journal of Grid Computing, 171-200.

YU, J. AND BUYYA, R. 2007. Workflow Scheduling Algorithms for Grid Computing Computing and Distributed Systems Laboratory, The University of Melbourne, Australia.

YU, Z. AND SHI, W. 2007. An Adaptive Rescheduling Strategy for Grid Workflow Applications. In 2007 IEEE International Symposium on Parallel and Distributed Processing (IPDPS 2007), Long Beach, California USA, 115-122.

ZHUGE, H., CHEUNG, T. AND PUNG, H. 2001. A Timed Workflow Process Model, Journal of Systems and Software 55, 231-243.

Received August 2011; revised March 2012 and July 2012; accepted January 2013