Temporal Dependency based Checkpoint Selection for Dynamic Verification of Temporal Constraints in Scientific Workflow Systems*

JINJUN CHEN, YUN YANG
Swinburne University of Technology

In a scientific workflow system, a checkpoint selection strategy is used to select checkpoints along scientific workflow execution for verifying temporal constraints so that we can identify any temporal violations and handle them in time in order to ensure overall temporal correctness of the execution which is often essential for the usefulness of execution results. The problem of existing representative strategies is that they do not differentiate temporal constraints as once a checkpoint is selected, they verify all temporal constraints. However, such checkpoint does not need to be taken for those constraints whose consistency can be deduced from others. The corresponding verification of such constraints is consequently unnecessary and can severely impact overall temporal verification efficiency while the efficiency determines whether temporal violations can be identified quickly for handling in time. To address the problem, in this paper, we develop a new temporal dependency based checkpoint selection strategy which can select checkpoints according to different temporal constraints. With our strategy, the corresponding unnecessary verification can be avoided. The comparison and experimental simulation further demonstrate that our new strategy can improve the efficiency of overall temporal verification significantly over the existing representative strategies.

Categories and Subject Descriptors: D.2.4 [**Software Engineering**]: Software/Program Verification
General Terms: Algorithms, Design, Reliability, Theory, Verification
Additional Key Words and Phrases: Scientific workflows, temporal constraints, temporal verification, checkpoint selection

1. INTRODUCTION

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 achieving increasing attention from software engineering researchers [Ludäscher et al. 2006, Seces 2008]. It is responsible for modelling and executing large-scale sophisticated scientific workflows existing in a variety of complex computation and data intensive applications such as astrophysics, climate modelling and earthquake simulation [Abramson et al. 2005, Daisuke et al. 2007, Gil et al. 2007, Mandal et al. 2007, Taylor et al. 2007]. A scientific workflow normally contains a large number of computation and data intensive activities [Deelman and Chervenak 2008, Maechling et al. 2005, Oinn

This research is partly supported by Australian Research Council Discovery Project under grant No. DP0663841, and Australian Research Council Linkage Project under grant No. LP0990393.
Authors' addresses: J. Chen and Y. Yang, Faculty of Information and Communication Technologies, Swinburne University of Technology, PO Box 218, Hawthorn, Melbourne, Australia 3122; email: {jchen; yyang}@swin.edu.au.

* A preliminary version of this paper was published in the 30th International Conference on Software Engineering (ICSE2008), Leipzig, Germany, 10-18 May 2008. See Ref [Chen and Yang 2008b]. The work reported in this paper is a significant extension and generalisation for all types of temporal constraints.

et al. 2006, Prodan and Fahringer 2008]. One of the software engineering research issues in developing a scientific workflow system is temporal verification which is to identify any temporal violations in scientific workflow specifications and executions [Chen and Yang 2008a, Yu and Buyya 2005].

1.1 Temporal Constraints

In reality, a scientific workflow is normally time constrained [Brandic et al. 2008, Pandey and Buyya 2008] because temporal correctness, i.e. whether the scientific workflow can be completed on time, is essential to ensure the usefulness of its execution results. For example, an astrophysics scientific workflow for detecting the existence of gravitational wave in a signal channel is a time-critical real-time streaming data application [Daisuke et al. 2007]. Even a small delay in its completion can result in the omission of detection due to its real-time and streaming nature. Then, several more years may be needed for another wave to appear, but the worse consequence is that this would not be known since we are not aware of the omission due to the time delay. As a result, all completed and continuing costly computation including expensive use of supercomputing facilities for thousands of hours becomes useless and hence a big economical loss[1]. Taking the supercomputer hosted and funded by Swinburne University of Technology in Australia [SwinSuper 2009] as an example, the charge for a use of 24 hours is about US$20000. For the use of thousands of hours, we can imagine how much the economical loss could be. As such, temporal constraints should be set in scientific workflow specifications to enable the control and monitoring of temporal correctness during execution. The types of temporal constraints mainly include: upper bound, lower bound and fixed-time [Chen and Yang 2008b, Eder et al. 1999]. 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 than or equal to it. A fixed-time constraint at an activity is an absolute time value such as 6:00pm by which the activity must be completed.

Comparing the three types of temporal constraints, we can find that conceptually a lower bound constraint is symmetrical to an upper bound constraint while a fixed-time constraint is a special case of upper bound constraint. The reasons are as follows. For a lower bound constraint, we often check whether the duration between its start and end activities is greater than or equal to ($\geq$) its value while for an upper bound constraint, we often check whether the duration between its start and end activities is less than or equal to ($\leq$) its value. Therefore, they are symmetrical to each other. As for a fixed-time constraint, the first activity of a scientific workflow is actually its start activity. Hence, a fixed-time constraint can be viewed as a special upper bound constraint whose start activity is the first activity and whose end activity is the one at which the fixed-time constraint is. Nevertheless, an upper bound constraint is conceptually more general than a fixed-time constraint as its start activity can be an intermediate activity rather than the first activity. Besides, different upper bound constraints can have different start activities while all fixed-time constraints have the same start activity which is the first activity.

As such, in this paper, we focus on upper bound constraints only. The corresponding discussion and results can be symmetrically applied to lower bound constraints and adaptively simplified for fixed-time constraints.

[1] In fact, there is also a loss of scientific discovery of the gravitational wave. However, as it is not relevant to the scope of this paper, we do not discuss it further.

1.2 Temporal Verification and Checkpoint Selection

After upper bound constraints are set, temporal verification must be conducted so that we can identify any temporal violations and handle them in time in order to ensure overall temporal correctness. Temporal verification efficiency reflects whether a temporal violation can be identified quickly. Back to the astrophysics scientific workflow stated in Section 1.1, a very small time delay can result in that the whole costly computation involving expensive use of supercomputing facilities for thousands of hours becomes useless and consequently a big economical loss. Hence, temporal violations need to be detected as soon as possible so that corresponding handling can be triggered in time to remove them in order to guarantee the overall temporal correctness of scientific workflow execution. Therefore, temporal verification efficiency plays a critical role in ensuring the overall temporal correctness of scientific workflow execution which is essential for the usefulness of execution results and for saving the corresponding huge cost as shown in the above mentioned astrophysics example.

Along scientific workflow execution, conducting temporal verification at all activities is not efficient as we may not have to do so at some activities such as those that can be completed within allowed time intervals. So we need to figure out where to conduct the verification. The activities at which we conduct the verification are called ***checkpoints*** [Marjanovic and Orlowska 1999, Zhuge et al. 2001]. A strategy used to select checkpoints for conducting the verification is called a checkpoint selection strategy, denoted as *CSS* [Marjanovic and Orlowska 1999, Zhuge et al. 2001]. Some representative checkpoint selection strategies have been proposed by [Chen et al. 2004, Chen and Yang 2005a, Chen and Yang 2008a, Chen and Yang 2007, Eder et al. 1999, Marjanovic and Orlowska 1999, Zhuge et al. 2001], which are detailed in Section 3. However, their common problem is that they treat all upper bound constraints as a whole because once an activity point is selected as a checkpoint, it is for verifying all upper bound constraints. But for some constraints, their consistency can be deduced from others. Such constraints do not need to take any checkpoints. The verification of them is consequently unnecessary, which can severely impact overall temporal verification efficiency since there are normally a large number of upper bound constraints in a scientific workflow. To address the problem for significantly improving temporal verification efficiency, in this paper we develop a new temporal dependency based checkpoint selection strategy which can select checkpoints corresponding to different upper bound constraints. We first investigate temporal dependency between different upper bound constraints. In general, temporal dependency means that upper bound constraints are dependent on each other in terms of their setting and verification. By temporal dependency, we can identify those upper bound constraints whose consistency can be deduced from others. Then, based on temporal dependency, we present our new strategy. With our new strategy, those upper bound constraints whose consistency can be deduced from others will no longer take any checkpoints. Consequently, the verification of them can be avoided which is otherwise incurred by the existing representative strategies. The comparison and experimental simulation further demonstrate that our strategy can improve overall temporal verification efficiency significantly over the existing representative ones.

In [Chen and Yang 2008b], we have discussed the general idea of temporal dependency based checkpoint selection. However, that is for fixed-time constraints only. As stated in Section 1.1, different upper bound constraints can have different start activities in contrast that all fixed-time constraints have the same start activity. This leads to that a later upper bound constraint may not completely cover a previous upper bound constraint while a fixed-time constraint does cover all previous fixed-time constraints completely. Accordingly, the relationship between upper bound constraints contains several scenarios as presented in Section 4.1 while the relationship between fixed-time constraints contains

only one scenario, i.e. all fixed-time constraints are nested one after another. As a result, the temporal dependency between upper bound constraints is much more general and complicated than that between fixed-time constraints. The two types of temporal dependency become identical only when all upper bound constraints have the same start activity. As such, this paper is a significant extension and generalisation of [Chen and Yang 2008b] to cover more general and complicated temporal dependency between upper bound constraints and its influence on checkpoint selection.

1.3 Paper Organisation

The remainder of the paper is organised as follows. In Section 2, we summarise some time attributes of scientific workflows. In Section 3, we detail the related work and problem analysis. Then, in Section 4, we discuss temporal dependency between upper bound constraints. After that, in Section 5, we apply temporal dependency to checkpoint selection and propose our new checkpoint selection strategy. In Section 6, we perform a comprehensive comparison and experimental simulation to demonstrate that our strategy can improve overall temporal verification efficiency significantly than the existing representative ones. Finally in Section 7, we conclude our contributions and point out future work.

2. OVERVIEW OF TIMED SCIENTIFIC WORKFLOW REPRESENTATION

According to [Li et al. 2003; Marjanovic and Orlowska 1999], based on the directed network graph (DNG) concept, a scientific workflow can be represented as a DNG-based scientific workflow graph, where nodes correspond to activities and edges correspond to dependencies between activities. In [Li et al. 2003; Marjanovic and Orlowska 1999], the iterative structure is nested in an activity that has an exit condition defined for iterative purposes. Accordingly, the corresponding DNG-based scientific workflow graph is structurally acyclic[2]. Here we assume that a scientific workflow is well structured, i.e. there are no any structure errors such as deadlocks, livelocks, dead activities and so on. The structure verification is outside the scope of this paper and can be referred to some other references such as [Aalst 2003; Sadiq and Orlowska 2000].

2.1 Activity Time Attributes and Temporal Constraints

To represent activity time attributes in a scientific workflow, we borrow some concepts from [Chinn and Madey 2000; Eder et al. 1999; Marjanovic and Orlowska 1999] such as maximum, mean or minimum duration as a basis. We denote the i^{th} activity of a scientific workflow as a_i. For each a_i, we denote its maximum duration, mean duration, minimum duration, run-time start time, run-time end time and run-time completion duration as $D(a_i)$, $M(a_i)$, $d(a_i)$, $S(a_i)$, $E(a_i)$ and $R(a_i)$ respectively. If there is a path from a_i to a_j ($i<j$), we denote the maximum duration, mean duration and minimum duration between them as $D(a_i, a_j)$, $M(a_i, a_j)$ and $d(a_i, a_j)$ respectively. $M(a_i)$ indicates that statistically a_i can be completed around its mean duration. Other time attributes are self-explanatory. According to [Chinn and Madey 2000; Son and Kim 2001], $D(a_i)$, $M(a_i)$ and $d(a_i)$ can be obtained based on the past execution history. The past execution history covers the delay time incurred at a_i such as the setup delay, queuing delay, synchronisation delay, network latency and so on. The detailed discussion of $D(a_i)$, $M(a_i)$ and $d(a_i)$ is outside the scope of this paper and can be referred to [Chen and Yang 2005b; Eder et al. 1999; Marjanovic and Orlowska 1999]. For a specific execution of a_i, the delay time is included in $R(a_i)$. Normally, we have $d(a_i) \le M(a_i) \le D(a_i)$ and $d(a_i) \le R(a_i) \le D(a_i)$. If there is a path from a_i to a_j

[2] Refer to [Li et al. 2003; Marjanovic and Orlowska 1999] for more details.

($i<j$) and an upper bound constraint between them, we denote the upper bound constraint as $U(a_i, a_j)$ and its value as $u(a_i, a_j)$. For a series of upper bound constraints, we denote them as U_1, U_2, U_3 and so forth, their values as $u(U_1)$, $u(U_2)$, $u(U_3)$ and so forth.

For convenience of the discussion, we only consider one execution path in the acyclic DNG-based scientific workflow graph without losing generality. As to a selective or parallel structure, each branch is an execution path. Therefore, we can equally apply the results achieved in this paper to each branch directly. In overall terms, for a scientific workflow containing many parallel, selective and/or mixed structures, firstly, we treat each structure as an 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. Correspondingly, between a_i and a_j, $D(a_i, a_j)$ is equal to the sum of all activity maximum durations, $M(a_i, a_j)$ is equal to the sum of all activity mean durations, and $d(a_i, a_j)$ is equal to the sum of all activity minimum durations.

2.2 Temporal Consistency States

Besides the time attributes represented in Section 2.1, [Chen and Yang 2005b] identifies and defines four temporal consistency states which are based on [Eder et al. 1999]. They are SC (Strong Consistency), WC (Weak Consistency), WI (Weak Inconsistency) and SI (Strong Inconsistency). Since the checkpoint concept is related to run-time execution stage and the temporal dependency addressed in Section 4 is related to build-time stage, we only summarise the definitions for these two stages here. The definitions for run-time instantiation stage and the detailed discussion can be found in [Chen and Yang 2005b].

Definition 1. At build-time stage, $U(a_i, a_j)$ is said to be of:

1) SC if with $D(a_i, a_j) \leq u(a_i, a_j)$;
2) WC if $M(a_i, a_j) \leq u(a_i, a_j) < D(a_i, a_j)$;
3) WI if $d(a_i, a_j) \leq u(a_i, a_j) < M(a_i, a_j)$;
4) SI if $u(a_i, a_j) < d(a_i, a_j)$.

Definition 2. At run-time execution stage, at checkpoint a_p between a_i and a_j, $U(a_i, a_j)$ is said to be of:

1) SC if $R(a_i, a_p) + D(a_{p+1}, a_j) \leq u(a_i, a_j)$;
2) WC if $R(a_i, a_p) + M(a_{p+1}, a_j) \leq u(a_i, a_j) < R(a_i, a_p) + D(a_{p+1}, a_j)$;
3) WI if $R(a_i, a_p) + d(a_{p+1}, a_j) \leq u(a_i, a_j) < R(a_i, a_p) + M(a_{p+1}, a_j)$;
4) SI if $u(a_i, a_j) < R(a_i, a_p) + d(a_{p+1}, a_j)$.

Definition 2 actually mixes the duration prediction after the checkpoint, i.e. $D(a_{p+1}, a_j)$, $M(a_{p+1}, a_j)$ and $d(a_{p+1}, a_j)$, with actual completion duration obtained until the checkpoint, i.e. $R(a_i, a_p)$. For clarity, we further depict SC, WC, WI and SI in Figure 1.

According to [Chen and Yang 2005b], along scientific workflow execution, for SC, we do not need to do anything as the corresponding upper bound constraints can be kept. For WC, by utilising the possible time redundancy of succeeding activity execution, i.e. the time saved by the execution of each succeeding activity from its pre-set maximum duration, the corresponding upper bound constraints may still be kept. Specific methods for utilising the possible time redundancy can be found in [Chen and Yang 2005b]. For WI and SI, basically for most cases, the corresponding upper bound constraints cannot be kept. Consequently, the corresponding exception handling needs to be triggered to adjust them to SC or WC. Specific exception handling methods can be borrowed and adapted from [Hagen and Alonso 2000, Russell et al. 2006].

Since WI and SI are adjusted to SC or WC by their respective exception handling, along grid workflow execution, checkpoint selection actually focuses on selecting

checkpoints for verifying previous SC and WC upper bound constraints to check their current consistency.

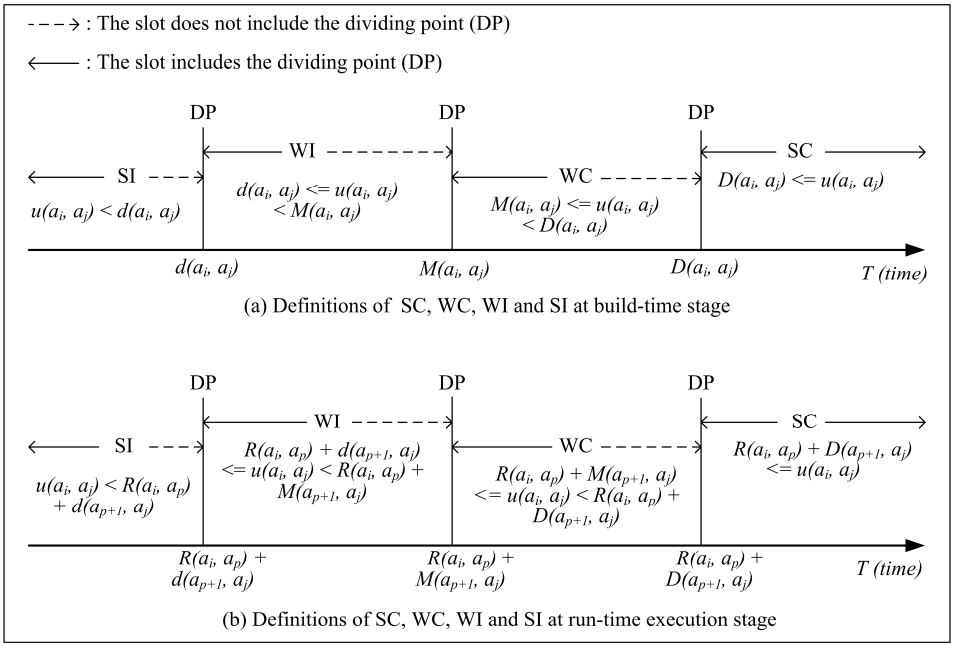

Fig. 1. Definitions of SC, WC, WI and SI at build-time and run-time execution stages

3. RELATED WORK AND PROBLEM ANALYSIS

Different representative checkpoint selection strategies have been proposed in the literature. To the best of our knowledge, we list them below.

- CSS_1: [Eder et al. 1999] takes every activity as a checkpoint. We denote this strategy as CSS_1.
- CSS_2: [Zhuge et al. 2001] sets checkpoints at the start time and end time of each activity. We denote this strategy as CSS_2.
- CSS_3: [Marjanovic and Orlowska 1999] takes the start activity as a checkpoint and adds a checkpoint after each decision activity is executed. We denote this strategy as CSS_3.
- CSS_4: [Marjanovic and Orlowska 1999] also mentions another checkpoint selection strategy: user-defined static checkpoints. That is that users define some static activity points as checkpoints at build-time stage. We denote this strategy as CSS_4.
- CSS_5: [Chen et al. 2004] selects activity a_i as a checkpoint if $R(a_i) > D(a_i)$. We denote this strategy as CSS_5.
- CSS_6: [Chen and Yang 2008a] selects activity a_i as a checkpoint if $R(a_i) > M(a_i)$. We denote this strategy as CSS_6.
- CSS_7: [Chen and Yang 2005a] introduces a minimum proportional time redundancy for each activity and then selects an activity as a checkpoint when its completion duration is greater than the sum of its mean duration and its minimum proportional time redundancy. We denote this strategy as CSS_7.
- CSS_8: [Chen and Yang 2007] introduces a minimum time redundancy for each activity and then selects an activity as a checkpoint when its completion duration

is greater than the sum of its mean duration and its minimum time redundancy. We denote this strategy as CSS_8.

All of $CSS_1 \sim CSS_7$ do not differentiate upper bound constraints. Once an activity point is selected as a checkpoint, they will verify all upper bound constraints. However, for some upper bound constraints, their consistency can be deduced from others. Such constraints do not need to take any checkpoints and do not need to be verified. CSS_8 can guarantee that at each selected checkpoint there is at least one upper bound constraint violated. Since it treats all upper bound constraints as a whole, it can claim that all checkpoints selected by it are "necessary" and "sufficient". However, those upper bound constraints whose consistency can be deduced from others do not need to take any checkpoints. That is to say, when we differentiate upper bound constraints, CSS_8 has a similar problem of $CSS_1 \sim CSS_7$.

We now consider the example of an astrophysics scientific workflow for detecting gravitational wave [Daisuke et al. 2007] for analysing the problem of $CSS_1 \sim CSS_8$. Such workflow can contain hundreds of thousands of activities and sub-activities such as computation resource discovery, data reduction and data transfer [Daisuke et al. 2007]. Depending on the detecting outcome, the workflow may execute for a few months or several years. For such a long time, astrophysics scientists often need to know intermediate execution results within various time periods during the execution so that they can decide on the subsequent actions. Accordingly, upper bound constraints are assigned for those periods so that corresponding intermediate results can be achieved on time. We consider three of them denoted as U_l, U_m and U_n. Although three only, they are sufficient for analysing the problem of $CSS_1 \sim CSS_8$. We denote the workflow segment covering U_l, U_m and U_n as the k^{th} segment and depict it in Figure 2. Some time values are also attached in Figure 2. The selection of those values is random and does not affect our analysis because the dependency between U_l, U_m and U_n reflects certain relative relationship between them, hence not subject to the absolute time values. Figure 2 contains a selective structure which has two branches, i.e. Branch 1 and Branch 2. We focus on SC of U_l, U_m and U_n. The corresponding discussion for WC is similar.

In Figure 2, we consider an execution instance where the execution goes Branch 1 and $R(a_i)$=8, $R(a_{i+1})$=15, $R(a_{i+2})$=19, $R(a_{i+3})$=16, $R(a_{i+4})$=14, $R(a_{i+5})$=9, $R(a_{i+6})$=4, $R(a_{i+7})$=5, $R(a_{i+8})$=15. Suppose a_{i+8} is selected as a checkpoint by one of $CSS_1 \sim CSS_8$. Then, at a_{i+8}, all of U_l, U_m and U_n will be verified according to Definition 2. We will find that U_l is not of SC, but U_m and U_n are of SC. However, we argue that U_n does not need to be verified because its consistency can be deduced from U_m. That is to say, U_n does not need to take a_{i+8} as a checkpoint. We explain as follows. $R(a_{i+4}, a_{i+8}) + D(a_{i+10}, a_{i+15}) = R(a_{i+4}) + R(a_{i+5}) + R(a_{i+6}) + R(a_{i+7}) + R(a_{i+8}) + D(a_{i+10}) + D(a_{i+11}) + D(a_{i+12}) + D(a_{i+13}) + D(a_{i+14}) + D(a_{i+15})$=14+9+4+5+15+6+8+15+20+18+12=126. We also have $u(U_m)$=150. Hence, we have inequation (1) below.

$$R(a_{i+4}, a_{i+8}) + D(a_{i+10}, a_{i+15}) < u(U_m) \quad (1)$$

a_{i+9} is not executed because it is on the other branch. Therefore, according to Definition 2, inequation (1) means that U_m is of SC. Meanwhile, we have $D(a_i, a_{i+3}) = D(a_i) + D(a_{i+1}) + D(a_{i+2}) + D(a_{i+3})$ =10+16+21+17=64, and $R(a_i, a_{i+3})=R(a_i) + R(a_{i+1}) + R(a_{i+2}) + R(a_{i+3})$= 8+15+19+ 16=58. Hence, we have $R(a_i, a_{i+3}) \leq D(a_i, a_{i+3})$. Besides, we also have $D(a_{i+16}, a_{i+17}) = D(a_{i+16})+D(a_{i+17})$=15+11=26. Accordingly, we have $D(a_i, a_{i+3}) + u(U_m) + D(a_{i+16}, a_{i+17})$ = 64+150+26=240 < $u(U_n)$=250. Therefore, we have inequation (2) below.

$$D(a_i, a_{i+3}) + u(U_m) + D(a_{i+16}, a_{i+17}) < u(U_n) \quad (2)$$

With inequation (2), we have $R(a_i, a_{i+8}) + D(a_{i+10}, a_{i+17}) = R(a_i, a_{i+3}) + R(a_{i+4}, a_{i+8}) + D(a_{i+10}, a_{i+15}) + D(a_{i+16}, a_{i+17}) < D(a_i, a_{i+3}) + u(U_m) + D(a_{i+16}, a_{i+17}) < u(U_n)$. Hence, we have inequation (3) below.

$$R(a_i, a_{i+8}) + D(a_{i+10}, a_{i+17}) < u(U_n) \quad (3)$$

According to Definition 2, inequation (3) means that U_n is of SC.

The above example has demonstrated that we do not need to verify U_n. We have actually deduced the consistency of U_n from U_m. Therefore, a_{i+8} is a real checkpoint for U_l and U_m but not for U_n. That is to say, we should differentiate upper bound constraints and select checkpoints corresponding to different upper bound constraints.

Considering the above example again, we can find that there is a key factor for us to deduce the consistency of U_n from U_m. That is inequation (2) which is exactly the essence of temporal dependency between U_m and U_n. We detail it in the next section.

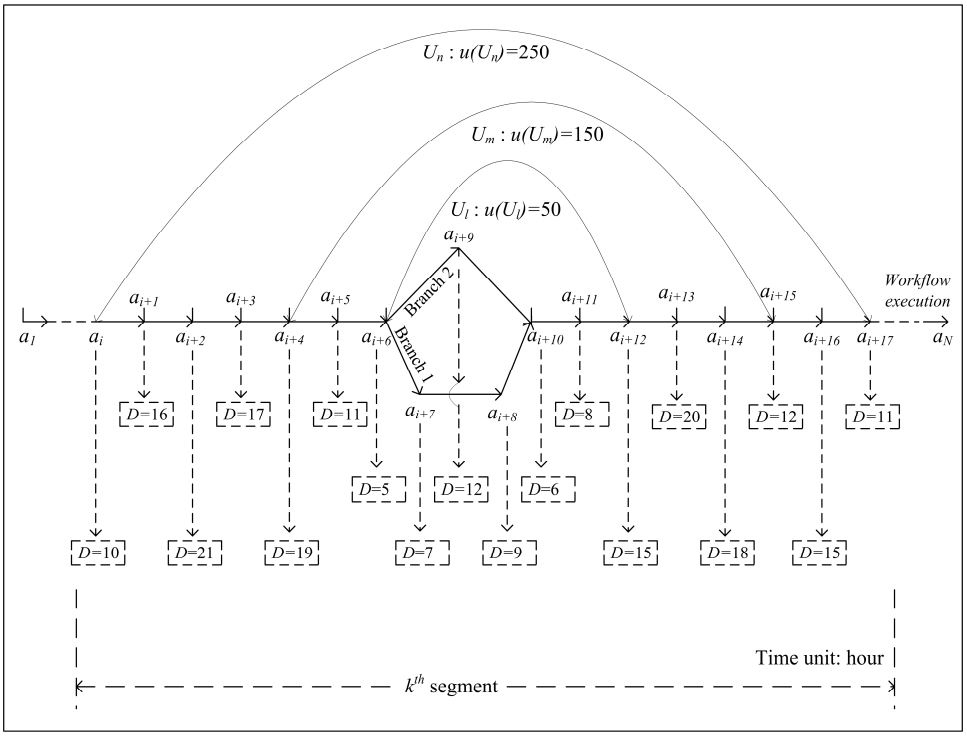

Fig. 2. An example of three upper bound constraints: U_l, U_m and U_n

4. TEMPORAL DEPENDENCY

In this section, we discuss temporal dependency between upper bound constraints. In general, temporal dependency means that different upper bound constraints are dependent on each other in terms of their setting and verification.

According to Section 2, since checkpoint selection is actually for SC and WC upper bound constraint verification, temporal dependency consists of SC temporal dependency and WC temporal dependency. The former is for SC upper bound constraints while the latter is for WC ones.

We first discuss SC and WC temporal dependency in Section 4.1. Then in Section 4.2, we investigate how to deduce the consistency of upper bound constraints based on temporal dependency.

4.1 SC and WC Temporal Dependency

We focus on SC temporal dependency. The discussion of WC temporal dependency is similar. We first discuss two upper bound constraints and then extend to multiple ones.

Considering two upper bound constraints U_1 and U_2 where U_1 is between a_{i_1} and a_{j_1}, and U_2 is between a_{i_2} and a_{j_2}, based on Allen's temporal interval logic [Allen 1983, Chinn and Madey 2000], we can conclude that there are two groups of basic relationships between U_1 and U_2. One is with non-nested relationship while the other is with nested relationship. We depict them in Figures 3 and 4 respectively. We are discussing temporal dependency generally between two upper bound constraints. Hence, we do not need to differentiate the ordering between U_1 and U_2.

In Figure 3, for Scenarios 1, 2, 4, 5 and 6, U_1 and U_2 are relatively independent as they do not have any activities in common. For Scenarios 3, 7 and 8, although U_1 and U_2 have some common activities, they still have many different activities which are independent of each other. Therefore, in Figure 3, U_1 and U_2 can be verified independently, i.e. no temporal dependency issue.

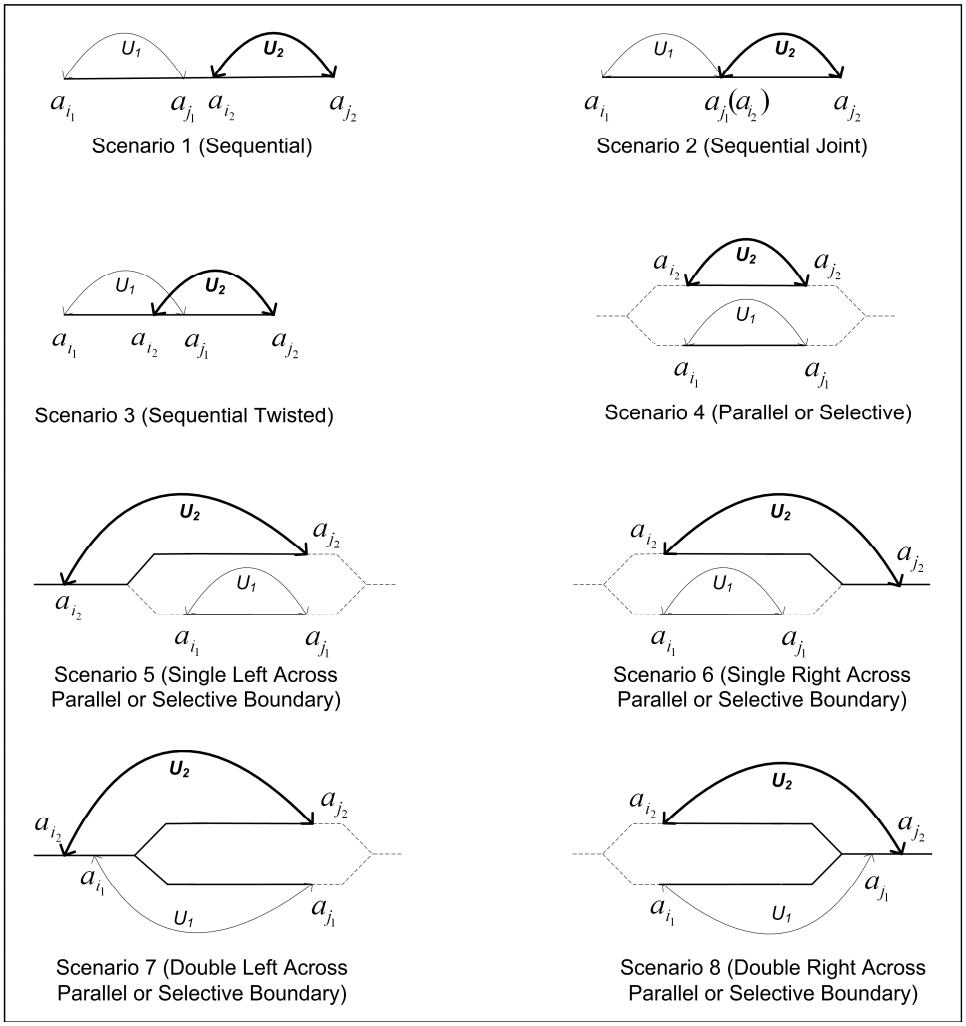

Fig. 3. Non-nested upper bound constraints U_1 and U_2

In Figure 4, for Scenario 9, if $u(U_2) \leq u(U_1)$, then, if U_2 is of SC, U_1 must be of SC. If U_2 is not of SC, we need to adjust U_2. The adjustment inevitably influences U_1 because

U_1 is included in U_2. Then, even if U_1 is of SC before the adjustment, we still need to re-verify it because its consistency may change after the adjustment. That is to say, the previous temporal verification of U_1 becomes useless. Therefore, in Scenario 9, we must ensure $u(U_1)<u(U_2)$. Similarly, in Scenarios 10 and 11, we must also ensure $u(U_1)<u(U_2)$.

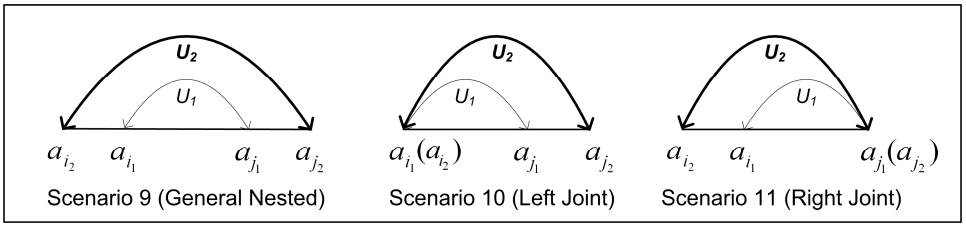

Fig. 4. Nested upper bound constraints U_1 and U_2

Now, we consider a more complicated situation for Scenario 9. Suppose we omit the temporal dependency between U_1 and U_2, and consequently set up them independently, we may encounter the following problem. We consider a special case where inequation (4) below holds.

$$u(U_2) < D(a_{i_2}, a_{i_1-1}) + u(U_1) + D(a_{j_1+1}, a_{j_2}) \quad (4)$$

With inequation (4), if U_2 is of SC, we have $D(a_{i_2}, a_{i_1-1}) + D(a_{i_1}, a_{j_1}) + D(a_{j_1+1}, a_{j_2}) \leq u(U_2) < D(a_{i_2}, a_{i_1-1}) + u(U_1) + D(a_{j_1+1}, a_{j_2})$. Then, we have inequation (5) below.

$$D(a_{i_1}, a_{j_1}) < u(U_1) \quad (5)$$

According to Definition 1, inequation (5) means that U_1 is of SC. Similar to the above situation where $u(U_2) \leq u(U_1)$, if U_2 is not of SC, even if U_1 is of SC, when we adjust U_2, we inevitably influence the setting of U_1 and need to re-verify U_1. Therefore, the previous temporal verification of U_1 becomes useless. Hence, in Scenario 9, $u(U_1) < u(U_2)$ is not enough and we still need to ensure that inequation (4) does not hold. That is to say, we need to make sure of inequation (6) below.

$$D(a_{i_2}, a_{i_1-1}) + u(U_1) + D(a_{j_1+1}, a_{j_2}) \leq u(U_2) \quad (6)$$

Similarly, in Scenario 10, we must ensure $u(U_1) + D(a_{j_1+1}, a_{j_2}) \leq u(U_2)$, and in Scenario 11, we must ensure $D(a_{i_2}, a_{i_1-1}) + u(U_1) \leq u(U_2)$.

In summary, temporal dependency between two upper bound constraints in Scenarios 9, 10 and 11 of Figure 4 must be taken into consideration in order to keep the previous temporal verification useful. Correspondingly, we have Definition 3 below.

Definition 3 (*SC Temporal Dependency*). Let U_1 and U_2 be two upper bound constraints (see 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} ($i_2 \leq i_1 < j_1 \leq j_2$), namely U_1 is nested in U_2. Then, with $D(a_{i_2}, a_{i_1-1}) + u(U_1) + D(a_{j_1+1}, a_{j_2}) \leq u(U_2)$, SC temporal dependency between U_1 and U_2 is defined as consistent.

For WC temporal dependency, similarly we have Definition 4 below.

Definition 4 (*WC Temporal Dependency*). Let U_1 and U_2 be two upper bound constraints (see 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} ($i_2 \leq i_1 < j_1 \leq j_2$), namely U_1 is nested in U_2. Then, with $M(a_{i_2}, a_{i_1-1}) + u(U_1) + M(a_{j_1+1}, a_{j_2}) \leq u(U_2) < D(a_{i_2}, a_{i_1-1}) + u(U_1) + D(a_{j_1+1}, a_{j_2})$, WC temporal dependency between U_1 and U_2 is defined as consistent.

We now investigate SC and WC temporal dependency between a series of upper bound constraints. In fact, based on the above discussion for two upper bound constraints,

we can see that if there are no nesting relationships like in Figure 3, there will be no temporal dependency issue. In reality, a scientific workflow normally has an end-to-end upper bound constraint which is a direct user requirement to cover from the start to the end of the workflow. Correspondingly, all other upper bound constraints are nested in it. That is to say, Figure 3 does not really reflect the practice of scientific workflow, but a deduced case from Allen's temporal interval logic for completeness. As such, we only need to consider the situation where a series of upper bound constraints are nested one after another. Considering N upper bound constraints U_1, U_2, ... , U_N, based on the nesting relationships between two upper bound constraints in Scenarios 9, 10 and 11 of Figure 4, we can derive four general groups of basic nesting relationships as depicted in Figure 5.

For the general case where N upper bound constraints are interleaved with each other, we can divide it into smaller groups according to their nesting relationships and then compose them from those scenarios in Figures 3, 4 and 5. The corresponding temporal dependency is composed as well. Hence, it is not a basic relationship and does not need to be discussed here.

In Figure 5, Scenario 12 is actually the extension of Scenario 9 of Figure 4. Scenario 13 is the extension of Scenario 10 of Figure 4. Scenario 14 is the extension of Scenario 11 of Figure 4. Scenario 15 is a combination of Scenarios 12, 13 and 14. Therefore, Figure 4 can be viewed as a special case of Figure 5. In addition, in reality, Figure 5 is also the most common case because we certainly need to consider other upper bound constraints when we set new ones. Setting upper bound constraints independently without any nesting relationships would easily cause potential conflict between them and confusion on the overall completion time of a scientific workflow execution. At least, all upper bound constraints are nested in the overall end-to-end one. As such, we focus on Figure 5 only to discuss temporal dependency.

To discuss temporal dependency in Scenarios 12, 13, 14 and 15 of Figure 5, we derive Theorem 1 next. Based on Theorem 1, the temporal dependency in Scenarios 12, 13, 14 and 15 can be translated into that in Scenarios 9, 10 and 11 of Figure 4 respectively.

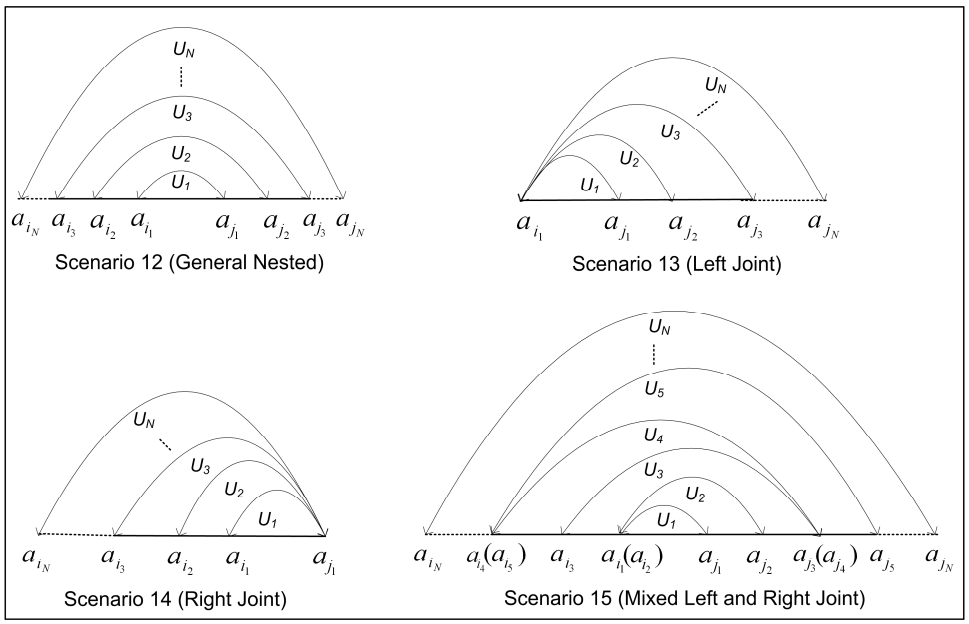

Fig. 5. A series of nested upper bound constraints

Theorem 1. Let U_1, U_2, ... , U_N be N upper bound constraints (see Figure 5) 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,

1) if SC temporal dependency between any two adjacent upper bound constraints U_k and U_{k+1} is consistent ($1 \le k \le N-1$), SC temporal dependency between any two non-adjacent upper bound constraints must also be consistent;
2) if WC temporal dependency between any two adjacent upper bound constraints U_k and U_{k+1} is consistent ($1 \le k \le N-1$), WC temporal dependency between any two non-adjacent upper bound constraints must also be consistent.

Proof: 1) We take Scenario 12 as the example to conduct the proof as Scenarios 13, 14 and 15 can actually be viewed as special cases of Scenario 12. For simplicity, we consider U_1, U_2 and U_3. Suppose SC temporal dependency between U_1 and U_2 is consistent and SC temporal dependency between U_2 and U_3 is consistent. Now we prove that SC temporal dependency between U_1 and U_3 is also consistent. That is to say, the consistency of SC temporal dependency is transitive. According to Definition 6, we have inequations (7) and (8) below.

$$D(a_{i_2}, a_{i_1-1}) + u(U_1) + D(a_{j_1+1}, a_{j_2}) \le u(U_2) \quad (7)$$

$$D(a_{i_3}, a_{i_2-1}) + u(U_2) + D(a_{j_2+1}, a_{j_3}) \le u(U_3) \quad (8)$$

Based on (7) and (8), we have: $D(a_{i_3}, a_{i_1-1}) + u(U_1) + D(a_{j_1+1}, a_{j_3}) = D(a_{i_3}, a_{i_2-1}) + D(a_{i_2}, a_{i_1-1}) + u(U_1) + D(a_{j_1+1}, a_{j_2}) + D(a_{j_2+1}, a_{j_3}) \le D(a_{i_3}, a_{i_2-1}) + u(U_2) + D(a_{j_2+1}, a_{j_3}) \le u(U_3)$. Hence, we have inequation (9) below.

$$D(a_{i_3}, a_{i_1-1}) + u(U_1) + D(a_{j_1+1}, a_{j_3}) \le u(U_3) \quad (9)$$

According to Definition 3, inequation (9) means that SC temporal dependency between U_1 and U_3 is consistent.

2) The proof is similar to 1), hence omitted.

Thus, in overall terms, the theorem holds. ▌

4.2 Consistency of Upper Bound Constraints

According to Section 4.1, at build-time stage we verify temporal dependency between any two adjacent nested upper bound constraints and accordingly make sure of its consistency. Then at run-time execution stage, we can derive Theorem 2 and Corollary 1 below. With them, we can deduce the consistency of later upper bound constraints from previous ones.

Scenario 12 of Figure 5 is more representative than other scenarios in Figures 4 and 5 as other scenarios can be viewed as special cases of it. Hence, we mainly focus on it. Correspondingly, Theorem 2 can be applied to all scenarios of Figures 4 and 5. For Scenario 13 of Figure 5 in particular where all upper bound constraints have the same start activity, Corollary 1 is deduced further by which we can directly deduce the consistency.

Theorem 2. Consider two upper bound constraints U_k and U_s ($k < s \le N$) (see Scenario 12 of Figure 5) where SC and WC temporal dependencies between U_k and U_s are consistent; U_k is between a_{i_k} and a_{j_k} and U_s is between a_{i_s} and a_{j_s} ($i_s < i_k < j_k < j_s$), i.e. U_k is nested in U_s. Then, at a_p between a_{i_k} and a_{j_k},

1) if $R(a_{i_s}, a_{i_k-1}) \le D(a_{i_s}, a_{i_k-1})$ and U_k is of SC, U_s must also be of SC.
2) if $R(a_{i_s}, a_{i_k-1}) \le M(a_{i_s}, a_{i_k-1})$ and U_k is of WC, U_s must also be of WC or even SC.

Proof: 1) If U_k is of SC, then with the consistency of temporal dependency between U_k and U_s, we have inequations (10) and (11) below.

$$R(a_{i_k}, a_p) + D(a_{p+1}, a_{j_k}) \le u(U_k) \quad (10)$$

$$D(a_{i_s}, a_{i_k-1}) + u(U_k) + D(a_{j_k+1}, a_{j_s}) \le u(U_s) \quad (11)$$

If $R(a_{i_s}, a_{i_k-1}) \le D(a_{i_s}, a_{i_k-1})$, based on inequations (10) and (11), we have $u(U_s) \ge D(a_{i_s}, a_{i_k-1}) + u(U_k) + D(a_{j_k+1}, a_{j_s}) \ge R(a_{i_s}, a_{i_k-1}) + u(U_k) + D(a_{j_k+1}, a_{j_s}) \ge R(a_{i_s}, a_{i_k-1}) + R(a_{i_k}, a_p) + D(a_{p+1}, a_{j_k}) + D(a_{j_k+1}, a_{j_s}) = R(a_{i_s}, a_p) + D(a_{p+1}, a_{j_s})$. Finally, we have inequation (12) below.

$$R(a_{i_s}, a_p) + D(a_{p+1}, a_{j_s}) \le u(U_s) \quad (12)$$

According to Definition 2, inequation (12) means that U_s is of SC.

2) The proof is similar to 1), hence omitted.

Thus, in overall terms, the theorem holds. ▌

Corollary 1. Let U_1, U_2, ... , U_N be N upper bound constraints (see Scenario 13 of Figure 5) where U_1 is between a_{i_1} and a_{j_1} and U_2 is between a_{i_1} and a_{j_2} and so forth ($i_1 < j_1 < j_2 < \ldots < j_N$), i.e. U_1, U_2, ... , U_N have the same start point a_{i_1} and U_1 is nested in U_2, U_2 is nested in U_3 and so forth. Then, at a_p between a_{i_1} and a_{j_k},

1) if U_k is of SC, any upper bound constraint U_s after U_k must also be of SC ($k < s \le N$).
2) if U_k is of WC, any upper bound constraint U_s after U_k must also be of WC or even SC ($k < s \le N$).

Proof: Since U_s and U_k have the same start point a_{i_1}, we have $R(a_{i_s}, a_{i_k-1}) = D(a_{i_s}, a_{i_k-1}) = M(a_{i_s}, a_{i_k-1}) = 0$. Hence, we always have $R(a_{i_s}, a_{i_k-1}) \le D(a_{i_s}, a_{i_k-1})$ and $R(a_{i_s}, a_{i_k-1}) \le M(a_{i_s}, a_{i_k-1})$. According to Theorem 2, the corollary holds. ▌

5. CHECKPOINT SELECTION BASED ON TEMPORAL DEPENDENCY

Among CSS_1~CSS_8, [Chen and Yang 2007] has experimentally demonstrated that CSS_8 can improve the checkpoint selection and eventual temporal verification efficiency significantly than other strategies. That is, CSS_8 is the best one among existing representative strategies. CSS_8 can guarantee that at each checkpoint there is at least one upper bound constraint violated. However, as analysed in Section 3.2, it does not differentiate upper bound constraints and will verify all of them at each checkpoint. In fact, a checkpoint is needed for some upper bound constraints, but not needed for those whose consistency can be deduced without further verification. As analysed in Section 4, with SC and WC temporal dependencies, we can derive such upper bound constraints. That is to say, with SC and WC temporal dependencies, we can overcome the problem of CSS_8. As such, we propose to facilitate SC and WC time redundancies to develop a new checkpoint selection strategy. We denote the new strategy as CSS_{TDB} (*TDB*: Temporal Dependency Based). In general, the working process of CSS_{TDB} is as follows. We first apply CSS_8 to determine whether an activity point is selected as a checkpoint for all upper bound constraints as a whole. Then, we apply SC and WC temporal dependencies to determine which upper bound constraints should take the checkpoint as a real one.

We now first summarise CSS_8 in Section 5.1. Then in Section 5.2, we present CSS_{TDB}.

5.1 Summary of CSS_8

CSS_8 introduces the concept of minimum time redundancy as a key judging parameter to determine whether an activity point is selected as a checkpoint. The minimum time

redundancy consists of minimum SC time redundancy and minimum WC time redundancy. In general, the idea of CSS_8 is as follows. Along workflow execution, at activity a_p, it computes the minimum SC and WC time redundancies. Then, it will determine whether the current time deviation is greater than the minimum SC or WC time redundancy. If so, there will be at least one SC or WC upper bound constraint violated. Then, CSS_8 selects a_p as a checkpoint, but for verifying all upper bound constraints.

5.1.1 Minimum Time Redundancies

At a_p, each SC upper bound constraint has a SC time redundancy defined in Definition 5. Each WC upper bound constraint has a WC time redundancy defined in Definition 6. Then, the minimum SC time redundancy at a_p is defined as the minimum one of all SC time redundancies while the minimum WC time redundancy at a_p is defined as the minimum one of all WC time redundancies. Definitions 7 and 8 below are for them respectively.

Definition 5. At activity point a_p between a_i and a_j ($i<j$), let $U(a_i, a_j)$ be of SC. Then, SC time redundancy of $U(a_i, a_j)$ at a_p is defined as $u(a_i, a_j) - [R(a_i, a_p) + D(a_{p+1}, a_j)]$ and denoted as $TR_{SC}(U(a_i, a_j), a_p)$: $TR_{SC}(U(a_i, a_j), a_p) = u(a_i, a_j) - [R(a_i, a_p) + D(a_{p+1}, a_j)]$.

Definition 6. At activity point a_p between a_k and a_l ($k<l$), let $U(a_k, a_l)$ be of WC. Then, WC time redundancy of $U(a_k, a_l)$ at a_p is defined as $u(a_k, a_l) - [R(a_k, a_p) + M(a_{p+1}, a_l)]$ and denoted as $TR_{WC}(U(a_k, a_l), a_p)$: $TR_{WC}(U(a_k, a_l), a_p) = u(a_k, a_l) - [R(a_k, a_p) + M(a_{p+1}, a_l)]$.

Definition 7 (*Minimum SC Time Redundancy*). Let $U_1, U_2, \ldots, U_N$ be N SC upper bound constraints and all of them cover a_p. Then, at a_p, the minimum SC time redundancy is defined as the minimum one of all SC time redundancies of $U_1, U_2, \ldots, U_N$, and is denoted as $MTR_{SC}(a_p)$ (MTR_{SC}: SC Minimum Time Redundancy):

$$MTR_{SC}(a_p) = Min\{ TR_{SC}(U_s, a_p) | s = 1,2, \ldots, N \}.$$

Definition 8 (*Minimum WC Time Redundancy*). Let $U_1, U_2, \ldots, U_N$ be N WC upper bound constraints and all of them cover a_p. Then, at a_p, the minimum WC time redundancy is defined as the minimum one of all WC time redundancies of $U_1, U_2, \ldots, U_N$, and is denoted as $MTR_{WC}(a_p)$ (MTR_{WC}: WC Minimum Time Redundancy):

$$MTR_{WC}(a_p) = Min\{ TR_{WC}(U_s, a_p) | s = 1,2, \ldots, N \}.$$

According to Definitions 7 and 8, at a_{p-1} or just before the execution of a_p, the minimum SC and WC time redundancies are $MTR_{SC}(a_{p-1})$ and $MTR_{WC}(a_{p-1})$ respectively.

5.1.2 Computation of Minimum Time Redundancies

[Chen and Yang 2007] developed a method named DOMTR (Dynamic Obtaining of Minimum Time Redundancy) for CSS_8 to dynamically compute $MTR_{SC}(a_p)$ and $MTR_{WC}(a_p)$ for each a_p. In general, DOMTR works as follows. Along scientific workflow execution, at a_p, DOMTR uses the time deviation caused by the execution of a_p to adjust previous minimum SC and WC time redundancies in order to achieve current ones. Specifically, $MTR_{SC}(a_p) = MTR_{SC}(a_{p-1}) - [R(a_p) - D(a_p)]$ and $MTR_{WC}(a_p) = MTR_{WC}(a_{p-1}) - [R(a_p) - M(a_p)]$. There are two exceptions for the computation as described next.

One is that a_p is the start activity of some SC and/or WC upper bound constraints. At such a_p, there are a SC minimum time difference and a WC minimum time difference assigned at build-time stage. An SC time difference of $U(a_p, a_j)$ at a_p is $u(a_p, a_j) - D(a_p, a_j)$. The SC minimum time difference is the minimum one of all SC time differences. A WC time difference of $U(a_p, a_l)$ at a_p is $D(a_p, a_l) - M(a_p, a_l)$. The WC minimum time difference is the minimum one of all WC time differences. For this situation, $MTR_{SC}(a_p) = Min\{MTR_{SC}(a_{p-1}),$ SC minimum time difference$\} - [R(a_p) - D(a_p)]$ and $MTR_{WC}(a_p) = Min\{MTR_{WC}(a_{p-1}),$ WC minimum time difference$\} - [R(a_p) - M(a_p)]$.

The other is when a_p is the end activity of the upper bound constraint of $MTR_{SC}(a_{p-1})$ or $MTR_{WC}(a_{p-1})$. In this case, $MTR_{SC}(a_{p-1})$ or $MTR_{WC}(a_{p-1})$ will become invalid after the execution of a_p because their upper bound constraint will be finished. Hence, we cannot use them to compute $MTR_{SC}(a_p)$ and $MTR_{WC}(a_p)$. In this case, DOMTR assigns the minimum one of all remaining SC time redundancies to $MTR_{SC}(a_p)$, and the minimum one of all remaining WC time redundancies to $MTR_{WC}(a_p)$.

The SC minimum time difference, WC minimum time difference, minimum one of all remaining SC time redundancies, and minimum one of all remaining WC time redundancies are all initialised at build-time stage, i.e. before workflow execution.

5.1.3 Checkpoint Selection of CSS_8

[Chen and Yang 2007] concluded and demonstrated the relationships between minimum SC & WC time redundancies and SC, WC, WI & SI of upper bound constraints. Based on those relationships, it proposed CSS_8. We depict the relationships in Figure 6. In Figure 6, "previous" means before the execution of a_p. Then, at a_p, the following three conclusions can be drawn.

1) If $R(a_p) > D(a_p)+MTR_{SC}(a_{p-1})$, we have to verify all previous SC and WC upper bound constraints. There is at least one previous SC upper bound constraint which is violated and now is not of SC. It is exactly the one whose SC time redundancy at a_{p-1} is $MTR_{SC}(a_{p-1})$.
2) If $M(a_p) + MTR_{WC}(a_{p-1}) < R(a_p) \leq D(a_p) + MTR_{SC}(a_{p-1})$, we need not verify all previous SC upper bound constraints, only all previous WC ones. And there is at least one previous WC upper bound constraint which is violated and now is not of SC and WC. It is exactly the one whose WC time redundancy at a_{p-1} is $MTR_{WC}(a_{p-1})$.
3) If $R(a_p) \leq M(a_p) + MTR_{WC}(a_{p-1})$, we need not verify all previous SC upper bound constraints. As to previous WC ones, based on [Chen and Yang 2005b] we do not need to verify them either. In [Chen and Yang 2005b], a method has been developed to adjust the WC upper bound constraints so that they can still be kept as SC. [Chen and Yang 2007] has proved that after execution of a_p, the status of the previous WC upper bound constraints is changed closer to SC (can even be changed to SC). Therefore, if a previous WC upper bound constraint is still of WC after execution of a_p, we can still use the previous adjustment on it. Hence, we do not need to do anything further to it. That is to say, we do not need to verify it.

Based on the above three conclusions, CSS_8 can determine whether a_p is selected as a checkpoint. The approach is: *At activity a_p, if $R(a_p) > D(a_p)+MTR_{SC}(a_{p-1})$, we take it as a checkpoint for verifying SC, WC, WI & SI of all previous SC upper bound constraints, and for verifying WC, WI & SI of all previous WC upper bound constraints. If $M(a_p)+MTR_{WC}(a_{p-1}) < R(a_p) \leq D(a_p)+MTR_{SC}(a_{p-1})$, we take a_p as a checkpoint for verifying SC, WC, WI & SI of all previous WC only upper bound constraints. If $R(a_p) \leq M(a_p) + MTR_{WC}(a_{p-1})$, we do not take a_p as a checkpoint.*

The whole working process of CSS_8 is that it employs DOMTR to compute the minimum SC and WC time redundancies at an activity, and then applies the above approach to determine whether the activity is selected as a checkpoint. With CSS_8, at each checkpoint there is at least one upper bound constraint violated.

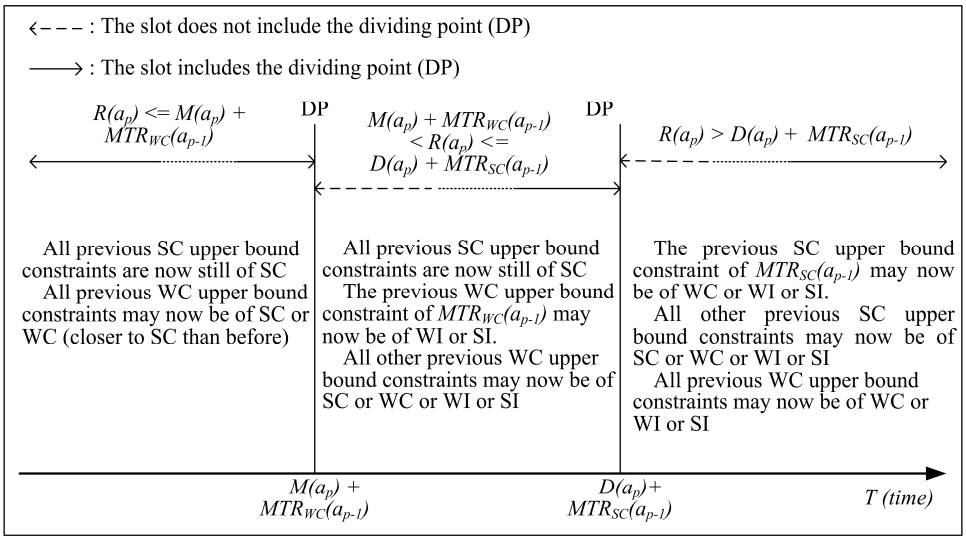

Fig. 6. Relationships between minimum SC and WC time redundancies and SC, WC, WI & SI

5.2 Checkpoint Selection Process of CSS_{TDB}

As stated earlier, CSS_{TDB} applies CSS_8 to determine whether the current activity point is selected as a checkpoint for all upper bound constraints as a whole. To be different from final real checkpoints, we call such a checkpoint as a tentative checkpoint. Then, CSS_{TDB} applies temporal dependency to figure out which upper bound constraints should take the tentative checkpoint as a real one. Based on Section 5.1 about CSS_8 and Section 4 about temporal dependency, we can derive the checkpoint selection process of CSS_{TDB}. The core part of it is depicted in Algorithm 1.

Input	Maximum, minimum and mean durations of all activities; ArraySC: an array of all SC upper bound constraints; ArrayWC: an array of all WC upper bound constraints.
Output	*True* or *False* as an appropriate checkpoint
Step 1	Compute $MTR_{SC}(a_p)$ and $MTR_{WC}(a_p)$ by DOMTR when scientific workflow execution arrives at a_p.
1.1. If a_p is not the start activity of any SC and WC upper bound constraints and a_p is not the end activity of the upper bound constraint of $MTR_{SC}(a_{p-1})$ or $MTR_{WC}(a_{p-1})$, then { $MTR_{SC}(a_p) = MTR_{SC}(a_{p-1}) - [R(a_p) - D(a_p)]$; $MTR_{WC}(a_p)=MTR_{WC}(a_{p-1}) - [R(a_p) - M(a_p)]$; }	
1.2. If a_p is the start activity of some SC and/or WC upper bound constraints, then { $MTR_{SC}(a_p)=Min\{MTR_{SC}(a_{p-1})$, SC minimum time difference$\}-[R(a_p)-D(a_p)]$; $MTR_{WC}(a_p)=Min\{MTR_{WC}(a_{p-1})$, WC minimum time difference$\}-[R(a_p)-M(a_p)]$; }	
1.3. If a_p is the end activity of the upper bound constraint of $MTR_{SC}(a_{p-1})$ or $MTR_{WC}(a_{p-1})$, then { $MTR_{SC}(a_p)$ = minimum one of all remaining SC time redundancy; $MTR_{WC}(a_p)$ = minimum one of all remaining WC time redundancy; }	
Step 2	Determine whether a_p is selected as a tentative checkpoint for all upper bound constraints as a whole.

2.1. If $R(a_p) > D(a_p)+MTR_{SC}(a_{p-1})$, then select a_p as a tentative checkpoint for all previous SC and WC upper bound constraints. 2.2. If $M(a_p) + MTR_{WC}(a_{p-1}) < R(a_p) \leq D(a_p) + MTR_{SC}(a_{p-1})$, then select a_p as a tentative checkpoint for all previous WC upper bound constraints. 2.3. If $R(a_p) \leq M(a_p) + MTR_{WC}(a_{p-1})$, then do not select a_p as a tentative checkpoint.	
Step 3	If a_p is already selected as a tentative checkpoint, figure out which upper bound constraints should not take it as a real checkpoint while the remaining ones should.
3.1. At a_p, verify upper bound constraints until an SC or WC one, say U_k is between a_{i_k} and a_{j_k}. Then, by Theorem 2, any upper bound constraint U_s (between a_{i_s} and a_{j_s}) nesting U_k with $R(a_{i_s}, a_{i_k-1}) \leq D(a_{i_s}, a_{i_k-1})$ or $R(a_{i_s}, a_{i_k-1}) \leq M(a_{i_s}, a_{i_k-1})$, does not take a_p as a real checkpoint. In addition, by Corollary 1, any upper bound constraint U_r having the same start activity of U_k does not take a_p as a real checkpoint.	

Algorithm 1. Checkpoint selection process of CSS_{TDB}

6. COMPARISON AND SIMULATION

6.1 Overall Comparison

As analysed in Section 3, existing representative checkpoint selection strategies do not differentiate upper bound constraints. Each checkpoint is for verifying all upper bound constraints. This will cause some unnecessary temporal verification because we do not need to verify those upper bound constraints whose consistency can be deduced without further verification. According to Section 5, CSS_{TDB} uses temporal dependency to derive the consistency of later upper bound constraints from previous ones. By this, CSS_{TDB} can identify those upper bound constraints whose consistency can be deduced without further verification. These upper bound constraints do not need to take the current tentative checkpoint as a real one. Consequently, their verification can be avoided which is currently incurred by the existing representative strategies. Therefore, with CSS_{TDB}, we can achieve better temporal verification efficiency.

In terms of the extra computation incurred by temporal dependency checking, it is only one or two additions at each activity covered by upper bound constraints. This, according to Definition 2, is actually equivalent to the computation for one-time temporal verification of one upper bound constraint. Since we normally need to conduct temporal verification many times at various activities for many upper bound constraints [Chen and Yang 2007; Marjanovic and Orlowska 1999; Zhuge et al. 2001], such one or two additions would be negligible.

6.2 Simulation

In this section, we perform an experimental simulation in our scientific workflow management system called SwinDeW-G (**Swin**burne **De**centralised **W**orkflow for **G**rid) [SwinDeW-G 2008, Yan et al. 2006]. The aim is to simulate temporal verification based on CSS_8 and CSS_{TDB} at run-time execution stage in order to demonstrate that CSS_{TDB} can improve temporal verification efficiency significantly than CSS_8. [Chen and Yang 2007] has experimentally demonstrated that CSS_8 can improve temporal verification efficiency significantly than other existing representative strategies. Therefore, if our aim is achieved, we are able to conclude that CSS_{TDB} can improve temporal verification efficiency significantly over all existing representative strategies.

In Section 6.2.1, we describe the simulation environment first. We then detail the simulation process in Section 6.2.2. In Section 6.2.3, we analyse the simulation results to demonstrate the significant improvement of CSS_{TDB} on temporal verification efficiency over CSS_8.

6.2.1 Simulation Environment

The key component in our simulation environment is SwinDeW-G which is running on a grid infrastructure named SwinGrid (Swinburne Grid) [SwinDeW-G 2008]. An overall picture of SwinGrid is depicted in the bottom plane of Figure 7 which contains many grid nodes distributed in different places. Each grid node contains many computers including high performance PCs and/or supercomputers composed of many computing units. The primary hosting nodes include the Swinburne CS3 (Centre for Complex Software Systems and Services) Node, Swinburne ESR (Enterprise Systems Research laboratory) Node, Swinburne Astrophysics Supercomputer Node, and Beihang CROWN Node in China. They are running Linux, GT4 (Globus Toolkit) or CROWN grid toolkits 2.5 [CROWN 2008, SwinDeW-G 2008] where CROWN (China R&D Environment Over Wide-area Network) is an extension of GT4 with more middleware, hence compatible with GT4. Besides, the CROWN Node is also connected to some other nodes such as those in Hong Kong University of Science and Technology, and University of Leeds in UK. The Swinburne Astrophysics Supercomputer Node is cooperating with PfC (Australian Platforms for Collaboration) and VPAC (Victorian Partnership for Advanced Computing).

Currently, SwinDeW-G is deployed at all primary hosting nodes. SwinDeW-G is a peer-to-peer based scientific workflow software system running on the SwinGrid infrastructure. A scientific workflow is executed by different peers that can be distributed at different grid nodes. Different peers communicate with each other directly in a peer-to-peer fashion. As shown in the bottom plane of Figure 7, each grid node can have a number of peers. In the top plane of Figure 7, we show a sample of how a scientific workflow can be executed in the simulation environment.

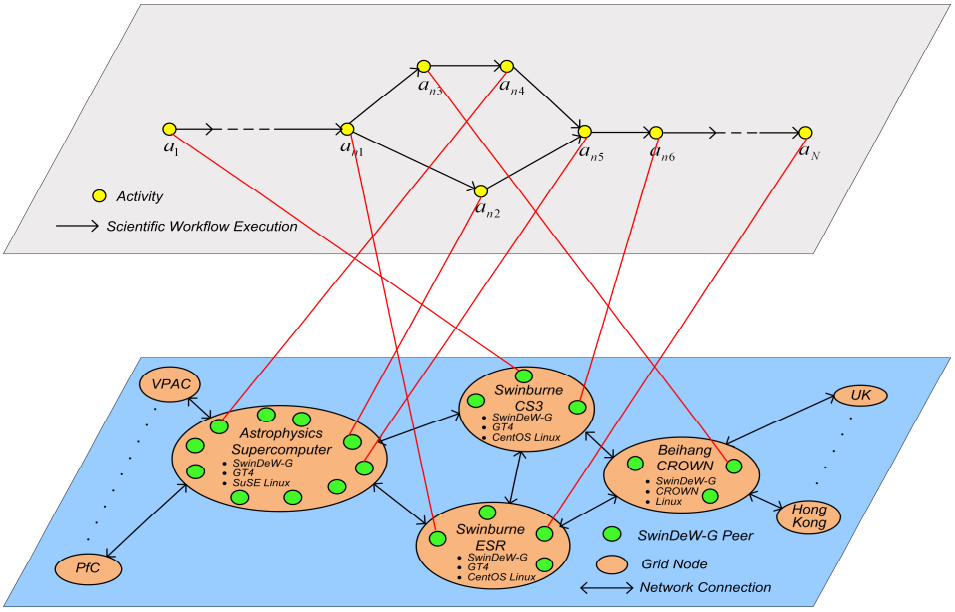

Fig. 7. Overview of the Simulation Environment

6.2.2 Simulation Process

We have simulated temporal verification on the astrophysics scientific workflow stated in Section 3 with CSS_{TDB} and CSS_8. Upper bound constraints in it are as shown in Scenario 12 of Figure 5. We chose Scenario 12 for simulation because it is more representative than other scenarios as stated in Section 4.2. The simulation process consists of two sub-processes detailed below. According to the definitions of temporal consistency in Section 2, the primary temporal verification computation is focused on the sum of maximum or mean durations between two activities. Therefore, we take each maximum or mean duration addition operation as a verification computation unit. Correspondingly, we perform the two simulation sub-processes in terms of the number of such units.

The first sub-process is as follows. Along scientific workflow execution, it executes CSS_8 to choose checkpoints. Then, at each checkpoint, it verifies all upper bound constraints. After it finishes all checkpoints and all upper bound constraints, we will have the number of all verification computation units. We denote it as $V(CSS_8)$.

The second sub-process is in parallel with the first one. It executes our new strategy CSS_{TDB} to choose appropriate checkpoints. Then, at each checkpoint, it verifies those upper bound constraints which take the checkpoint as a real one. After it finishes all checkpoints and corresponding upper bound constraints, we will have another number of all verification units. We denote it as $V(CSS_{TDB})$.

By comparing $V(CSS_{TDB})$ with $V(CSS_8)$, we are able to identify the significant improvement on temporal verification efficiency by CSS_{TDB} over CSS_8.

6.2.3 Simulation Results and Analysis

Based on the simulation process described in Section 6.2.2, we can derive $V(CSS_{TDB})$ and $V(CSS_8)$. They, together with corresponding trajectories, are depicted in Figure 8. They change by the number of upper bound constraints.

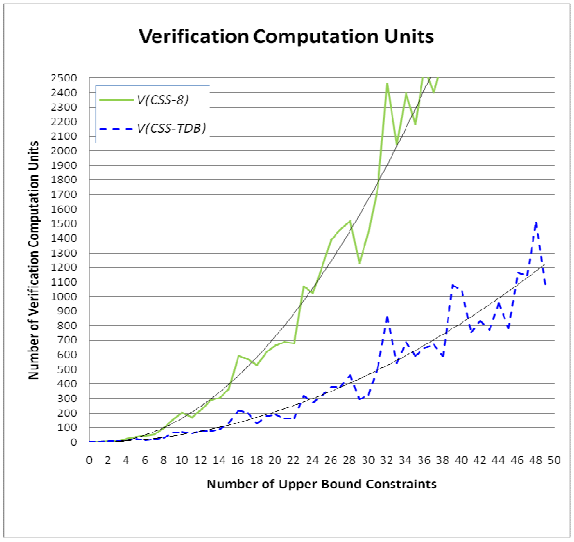

Fig. 8. Verification Computation Units by CSS_{TDB} and CSS_8

In Figure 8, we can see the overall trend being that with the number of upper bound constraints increasing, both $V(CSS_{TDB})$ and $V(CSS_8)$ increase. Locally, we can see the jitter on both curves of $V(CSS_{TDB})$ and $V(CSS_8)$. This is because scientific workflow execution is

very dynamic. Sometimes, the execution environment may be under a smooth condition with all required resources always available for executing scientific workflow activities. The characteristic of the execution environment is that there are almost no system failures and most workflow activities can be completed on time. Under such a circumstance, even more upper bound constraints may have fewer temporal violations and hence less verification computation than the case where there are fewer upper bound constraints, but the execution environment is under a less smooth condition whose characteristic is that there are a number of system failures and many workflow activities can not be completed on time. Then the jitter occurs. Nevertheless, this is only to some extent as the execution environment should be stable in overall terms although it may fluctuate locally. Otherwise, it needs to be improved in the first place before it can be used. Hence, the overall trend is still that the more upper bound constraints, the more violations and hence more verification computation, i.e. more $V(CSS_{TDB})$ and $V(CSS_8)$, will be caused. Furthermore, we can see that $V(CSS_8)$ goes up dramatically while $V(CSS_{TDB})$ rises relatively slower. In particular, when the number of upper bound constraints is getting larger, $V(CSS_8)$ is getting much greater than $V(CSS_{TDB})$. That is, the more upper bound constraints, the more significant improvement on verification efficiency by CSS_{TDB} over CSS_8. Since a scientific workflow normally contains hundreds of thousands of activities and lasts a long time, a large number of upper bound constraints are often needed so that the corresponding workflow execution can be monitored at various activities in order to ensure overall temporal correctness [Chen and Yang 2008a, Maechling et al. 2005]. That is to say, the real world situation is on the far right-hand side along the X axis. Therefore, we can conclude that with CSS_{TDB}, we can improve temporal verification efficiency significantly over CSS_8.

In addition, [Chen and Yang 2007] has experimentally demonstrated that CSS_8 can improve checkpoint selection and eventually temporal verification efficiency significantly over other existing representative strategies $CSS_1 \sim CSS_7$. Therefore, in overall terms, we can conclude that with our new checkpoint selection strategy CSS_{TDB}, we can improve temporal verification efficiency significantly over all existing representative strategies $CSS_1 \sim CSS_8$.

7. CONCLUSIONS AND FUTURE WORK

In a scientific workflow system, a checkpoint selection strategy is used to select checkpoints along scientific workflow execution for verifying temporal constraints so that we can identify any temporal violations and handle them in time in order to ensure overall temporal correctness of the execution which is essential for the usefulness of workflow execution results. However, this is a complex issue. The problem of existing representative checkpoint selection strategies is that they do not differentiate temporal constraints since a checkpoint is always selected for verifying all temporal constraints including those whose consistency can be deduced from others. Such constraints actually do not need to take any checkpoints. Consequently, the verification of them is unnecessary, which can severely impact overall temporal verification efficiency since there are normally a large number of temporal constraints in a scientific workflow. The temporal verification efficiency reflects whether a temporal violation can be identified quickly while temporal violations should be detected as soon as possible so that corresponding handling can be triggered in time to remove them in order to guarantee the overall temporal correctness of scientific workflow execution. As such, temporal verification efficiency plays a critical role in ensuring the overall temporal correctness of scientific workflow execution.

To address the above problem, in this paper, by taking upper bound constraint as the example, we have developed a new checkpoint selection strategy named CSS_{TDB} (Temporal Dependency Based Checkpoint Selection Strategy). As analysed in this paper, CSS_{TDB} can be symmetrically applied to lower bound constraints and adaptively

simplified for fixed-time constraints, hence applicable to all types of temporal constraints. CSS_{TDB} can make checkpoint selection corresponding to different temporal constraints. Specifically, temporal dependency between temporal constraints has been identified and investigated comprehensively. With temporal dependency, the consistency of some later temporal constraints can be deduced from previous ones. Then, based on temporal dependency, CSS_{TDB} was presented. With CSS_{TDB}, those later temporal constraints whose consistency can be deduced from previous ones will no longer take any checkpoints. Accordingly, their verification can be avoided which is otherwise incurred by existing representative strategies. The final comprehensive comparison and experimental simulation have shown that compared to existing representative strategies, CSS_{TDB} can significantly improve overall temporal verification efficiency.

With these contributions, we can further investigate how to handle temporal violations identified at a checkpoint such as how to compensate the time deficit dynamically along scientific workflow execution.

ACKNOWLEDGMENTS

The authors are grateful for the constructive comments of the anonymous reviewers, the simulation work of S. Hunter, the support from the CROWN team and the Swinburne Astrophysics and Supercomputing group, and the English proofreading by G. Foster.

REFERENCES

AALST, W.M.P. VAN DER, HOFSTEDE, A.H.M. TER, KIEPUSZEWSKI, B., AND BARROS, A.P. 2003. Workflow Patterns. *Distributed and Parallel* Databases *14(1)*, 5-51.

ABRAMSON, D., KOMMINENI, J., MCGREGOR, J.L., AND KATZFEY, J. 2005. An Atmospheric Sciences Workflow and its Implementation with Web Services. *Future Generation Computer Systems 21(1)*, 69-78.

ALLEN, J.F. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM 26(11)*, 832-843.

BRANDIC, I., PLLANA, S., AND BENKNER, S. 2008. Specification, Planning, and Execution of QoS-aware Grid Workflows within the Amadeus Environment. *Concurrency and Computation: Practice and Experience 20(4)*, 331-345.

CHEN, J., YANG, Y., AND CHEN, T. Y. 2004. Dynamic Verification of Temporal Constraints on-the-fly for Workflow Systems. In *Proceedings of the 11th Asia-Pacific Software Engineering Conference*, Busan, Korea, Nov./ Dec. 2004, IEEE CS Press, 30-37.

CHEN, J., AND YANG, Y. 2005a. A Minimum Proportional Time Redundancy based Checkpoint Selection Strategy for Dynamic Verification of Fixed-time Constraints in Grid Workflow Systems. In *Proceedings of the 12th Asia-Pacific Software Engineering Conference*, Taipei, Taiwan, Dec. 2005, IEEE CS Press, 299-306.

CHEN, J., AND YANG, Y. 2005b. Multiple Temporal Consistency States for Dynamical Verification of Upper Bound Constraints in Grid Workflow Systems. *In Proceedings of the 1st IEEE International Conference on e-Science and Grid Computing*, Melbourne, Australia, Dec. 2005, IEEE CS Press, 124-131.

CHEN, J., AND YANG, Y. 2008a. Activity Completion Duration based Checkpoint Selection for Dynamic Verification of Temporal Constraints in Grid Workflow Systems. *International Journal of High Performance Computing Applications 22(3)*, 319-329.

CHEN, J., AND YANG, Y. 2007. 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(2)*, Article 6.

CHEN, J., AND YANG, Y. 2008b. Temporal Dependency based Checkpoint Selection for Dynamic Verification of Fixed-time Constraints in Grid Workflow Systems. In *Proceedings of the 30th International Conference on Software Engineering (ICSE2008)*, Leipzig, Germany, May 2008, 141-150.

CHINN, S., AND MADEY, G. 2000. Temporal Representation and Reasoning for Workflow in Engineering Design Change Review. *IEEE Transactions on Engineering Management 47(4)*, 485-492.

CROWN Team. 2008. CROWN portal, http://www.crown.org.cn/en/, accessed on June 10, 2009.

DAISUKE, K., RUNYUE, C., LUIS, C.H. 2007. Gravitational Stability of Circumnuclear Disks in Elliptical Galaxies. *The Astrophysical Journal 669(1)*, 232-240.

DEELMAN, E., CHERVENAK, A.L. 2008. Data Management Challenges of Data-Intensive Scientific Workflows. In *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid*, May 2008, Lyon, France, IEEE CS Press, 687-692.

EDER, J., PANAGOS, E., AND RABINOVICH, M. 1999. Time Constraints in Workflow Systems. In *Proceedings of the 11th International Conference on Advanced Information Systems Engineering*, Heidelberg, Germany, June 1999, Springer Verlag, LNCS 1626, 286-300.

GIL, Y., DEELMAN, E., ELLISMAN, M., FAHRINGER, T., FOX, G., GANNON D., GOBLE C., LIVNY M., MOREAU L., AND MYERS J. 2007, Examining the Challenges of Scientific Workflows. *IEEE Computer 40(12)*, 24-32.

HAGEN, C., AND ALONSO, G. 2000. Exception Handling in Workflow Management Systems. *IEEE Transactions on Software Engineering 26(10)*, 943-958.

LI, J., FAN, Y., AND ZHOU, M. 2003. Timing Constraint Workflow Nets for Workflow Analysis. *IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans 33(2)*, 179-193.

LUDÄSCHER, B., ALTINTAS, I., BERKLEY, C., HIGGINS, D., JAEGER-FRANK, E., JONES, M., LEE, E., TAO, J., AND ZHAO, Y. 2006. Scientific Workflow Management and the Kepler System. *Concurrency and Computation: Practice and Experience 18(10)*, 1039-1065.

MAECHLING, P., CHALUPSKY, H., DOUGHERTY, M., DEELMAN, E., GIL, Y., GULLAPALLI, S., GUPTA, V., KESSELMAN, C., KIM, J., MEHTA, G., MENDENHALL, B., RUSS, T., SINGH, G., SPRARAGEN, M., STAPLES, G., AND VAHI, K. 2005, Simplifying Construction of Complex Workflows for Non-expert Users of the Southern California Earthquake Center Community Modeling Environment. *ACM SIGMOD Record 34(3)*, 24-30.

MANDAL, N., DEELMAN, E., MEHTA, G., SU, M., AND VAHI, K. 2007. Integrating Existing Scientific Workflow Systems: The Kepler/Pegasus Example. In *Proceedings of the 16th IEEE International Symposium on High Performance Distributed Computing*, Monterrey, CA, June 2007, 21-28.

MARJANOVIC, O., AND ORLOWSKA, M. E. 1999. On Modeling and Verification of Temporal Constraints in Production Workflows. *Knowledge and Information Systems 1(2)*, 157-192.

OINN, T., GREENWOOD, M., ADDIS, M., ALPDEMIR, M.N., FERRIS, J., GLOVER, K., GOBLE, C., GODERIS, A., HULL, D., MARVIN, D., LI, P., LORD, P., POCOCK, M.R., SENGER, M., STEVENS, R., WIPAT, A., WROE, C. 2006. Taverna: Lessons in Creating a Workflow Environment for the Life Sciences. *Concurrency and Computation: Practice and Experience 18(10)*, 1067-1100.

PANDEY, S., AND BUYYA, R. 2008. Scheduling of Scientific Workflows on Data Grids. In *Proceedings of the 8th IEEE International Symposium on Cluster Computing and the Grid*, May 2008, Lyon, France, IEEE CS Press, 548-553.

PRODAN, R. AND FAHRINGER, T. 2008. Overhead Analysis of Scientific Workflows in Grid Environments, *IEEE Transactions on Parallel and Distributed Systems 19(3)*, 378-393.

RUSSELL, N., AALST, W.M.P. VAN DER., AND HOFSTEDE, A.H.M. TER. 2006. Workflow Exception Patterns. In *Proceedings of the 18th International Conference on Advanced Information Systems Engineering*, Springer-Verlag, Berlin, 2006, LNCS 4001, 288-302.

SADIQ, W., AND ORLOWSKA, M.E. 2000. Analysing Process Models using Graph Reduction Techniques. *Information Systems 25(2)*, 117-134.

SECES, 2008, First International Workshop on Software Engineering for Computational Science and Engineering, in conjunction with the 30th International Conference on Software Engineering (ICSE2008), Leipzig, Germany, May 2008, http://www.cs.ua.edu/~SECSE08/, accessed on June 10, 2009.

SON, J. H., AND KIM, M. H. 2001. Improving the Performance of Time-constrained Workflow Processing. *The Journal of Systems and Software 58(3)*, 211-219.

SWINDEW-G Team. 2008. System Architecture of SwinDeW-G. http://www.swinflow.org/docs/System_Architecture.pdf, accessed on June 10, 2009.

SWINSUPER. 2009. http://astronomy.swin.edu.au/supercomputing/, accessed on June 10, 2009.

TAYLOR, I., SHIELDS, M., WANG, I., AND HARRISON, A. 2007. The Triana Workflow Environment: Architecture and Applications. In Ian Taylor, Deelman, E., Gannon, D., and Shields, M., editors, *Workflows for e- Science*. Springer, New York, Secaucus, NJ, USA, 2007, 320–339.

YAN, J., YANG, Y., AND RAIKUNDALIA, G.K. 2006. SwinDeW - A Peer-to-Peer based Decentralized Workflow Management System. IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans, IEEE Transactions on Systems, Man and Cybernetics - Part A 36(5), 922-935.

YU, J., AND BUYYA, R. 2005, A Taxonomy of Scientific Workflow Systems for Grid Computing, *ACM SIGMOD Record 34(3)*, 44-49.

ZHUGE, H., CHEUNG, T., AND PUNG, H. 2001. A Timed Workflow Process Model. *The Journal of Systems and Software 55(3)*, 231-243.