

Temporal Dependency based Checkpoint Selection for Dynamic Verification of Fixed-time Constraints in Grid Workflow Systems

Jinjun Chen, Yun Yang
Faculty of Information and Communication Technologies
Swinburne University of Technology
PO Box 218, Hawthorn, Melbourne, Australia 3122
+61 - 3 - 9214 8739
{jchen, yyang}@swin.edu.au

ABSTRACT

In grid workflow systems, temporal correctness is critical to assure the timely completion of grid workflow execution. To monitor and control the temporal correctness, fixed-time constraints are often assigned to a grid workflow and then verified. A checkpoint selection strategy is used to select checkpoints along grid workflow execution for verifying fixed-time constraints. The problem of existing representative strategies is that they do not differentiate fixed-time constraints as once a checkpoint is selected, they verify all fixed-time constraints. However, these checkpoints do 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 the efficiency of overall temporal verification. To address the problem, in this paper, we develop a new temporal dependency based checkpoint selection strategy which can select checkpoints according to different fixed-time 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 Term

Algorithms, Design, Reliability, Theory, Verification.

Keywords

Grid workflows, Fixed-time constraints, Temporal dependency, Checkpoint selection

1. INTRODUCTION

In a grid architecture, a grid workflow system is a type of high-level grid middleware. It is supposed to support large-scale sophisticated scientific or business processes in a variety of complex e-science or e-business applications such as climate modelling, disaster recovery, medical surgery, high energy physics, or international stock market modelling [22, 30, 35, 40]. Such sophisticated processes are modelled or redesigned as grid workflow specifications at build-time stage [15, 20, 21, 24]. The specifications normally contain a large number of computation and data intensive activities [2, 4, 35, 37]. Then, at run-time instantiation stage, grid workflow instances are created [8, 9, 27]. Finally, at run-time execution stage, grid workflow instances are executed by facilitating the super computing and data sharing ability of underlying grid infrastructure to complete the computation or data intensive activities [18, 23, 25, 31, 33].

In reality, complex scientific or business processes are normally time constrained [3, 5, 6, 7, 28] as temporal correctness is critical to assure the timely completion of their execution. Consequently, when they are modelled or redesigned as grid workflow specifications at build-time stage, fixed-time constraints are often set as well [2, 7, 19]. A fixed-time constraint at an activity is an absolute time value by which the activity must be completed [7, 19]. For example, a climate modelling grid workflow must be completed by a scheduled time [2], say 6:00pm, so that the weather forecast can be broadcast on time at a later time. Here, 6:00pm is a fixed-time constraint. When a grid workflow is executed, it starts from the first activity and then reaches each fixed-time constraint gradually. Correspondingly, we can view the first activity as the start point of each fixed-time constraint, i.e. all fixed-time constraints have the same start point [7, 19].

After fixed-time constraints are set, temporal verification must be conducted so that we can identify any temporal violations and consequently can take proper handling action in time. At build-time and run-time instantiation stages, temporal verification is static because there are no any specific execution times. Each fixed-time constraint needs only be verified once with the consideration of all covered activities. Therefore, we do not need to decide at which activities we should conduct the verification. At run-time execution stage however, activity completion durations vary and consequently, we may need to verify each fixed-time constraint many times at different activities. However, conducting the verification at every activity is not efficient as we may not have to do so at some activities such as those that can be completed within

Permission to make digital or hard copies of all or part 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 bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

ICSE '08, May 10–18, 2008, Leipzig, Germany.

Copyright 2008 ACM 978-1-60558-079-1/08/05...\$5.00.

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* [14, 32, 41]. A strategy used to select checkpoints for verifying fixed-time constraints is called a checkpoint selection strategy, denoted as CSS [14, 32, 41].

Some representative checkpoint selection strategies have been proposed by [10, 11, 13, 14, 19, 32, 41], which are detailed in Section 3. However, they treat all fixed-time constraints as a whole as once an activity point is selected as a checkpoint, it is for verifying all fixed-time constraints. But for some fixed-time 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 the efficiency of overall temporal verification since there are normally a large number of fixed-time constraints in a grid workflow. The efficiency of overall temporal verification is important because it directly affects whether current temporal verification is useful. As such, in this paper we develop a new temporal dependency based checkpoint selection strategy which can select checkpoints corresponding to different fixed-time constraints. We first investigate temporal dependency between different fixed-time constraints. In general, temporal dependency means that different fixed-time constraints are dependent on each other in terms of their setting and verification. By temporal dependency, we can identify those fixed-time constraints whose consistency can be deduced from others. Then, based on temporal dependency, we present our new strategy. With our new strategy, those fixed-time constraints whose consistency can be deduced from others will no longer take any checkpoints. Consequently, the verification of them can be avoided which is currently incurred by the existing representative strategies. The comparison and experimental simulation further demonstrate that our strategy can improve the efficiency of overall temporal verification significantly over the existing representative strategies.

The remainder of the paper is organised as follows. In Section 2, we summarize some time attributes of grid workflows. In Section 3, we detail the related work and problem analysis. Then, in Section 4, we discuss temporal dependency between fixed-time 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 strategies. Finally in Section 7, we conclude our contributions and point out future work.

2. OVERVIEW OF TIMED GRID WORKFLOW REPRESENTATION

According to [29, 34], based on the directed network graph (DNG) concept, a grid workflow can be represented as a DNG-based grid workflow graph, where nodes correspond to activities and edges correspond to dependencies between activities. In [29, 34], the iterative structure is nested in an activity that has an exit condition defined for iterative purposes. Accordingly, the corresponding DNG-based grid workflow graph is structurally acyclic¹. Here, we assume that the DNG-based grid workflow graph is well structured, i.e. there are no any structure errors such as deadlocks or dead

activities. The structure verification is outside the scope of this paper and can be referred to some other literature such as [1, 34].

To represent activity time attributes, we borrow some concepts from [12, 19, 32] such as maximum, mean or minimum duration as a basis. We denote the i^{th} activity of a grid workflow gw as a_i . We denote the expected time from which the specification of grid workflow gw will come into effect as $C(gw)$. From $C(gw)$, the specification of grid workflow gw can be used. 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. The mean duration $M(a_i)$ means that statistically a_i can be completed around its mean duration. Other time attributes are self-explanatory. According to [16, 36], $D(a_i)$, $M(a_i)$ 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 [12, 19, 32]. For a specific execution of a_i , the delay time is included in $R(a_i)$. Normally, we have $d(a_i) \leq M(a_i) \leq D(a_i)$ and $d(a_i) \leq R(a_i) \leq D(a_i)$. If there is a fixed-time constraint at a_i , we denote it as $F(a_i)$ and its value as $f(a_i)$. For a series of fixed-time constraints, we denote them as F_1, F_2, F_3 and so forth, their values as $f(F_1), f(F_2), f(F_3)$ and so forth. If there is a path from a_i to a_j ($i \leq j$), we denote the maximum duration, mean duration, minimum duration, run-time completion duration between them as $D(a_i, a_j)$, $M(a_i, a_j)$, $d(a_i, a_j)$ and $R(a_i, a_j)$ respectively [19, 32].

For convenience of the discussion, we focus on one execution path in the acyclic DNG-based grid 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 grid workflow containing many parallel, selective and/or mixed structures, firstly, we treat each structure as an activity. Then, the whole grid 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.

Besides the above time attributes, [12] has identified and defined four temporal consistency states which are based on [19]. 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 [12].

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

- (1) SC if $D(a_i, a_j) \leq f(a_j) - C(gw)$;
- (2) WC if $M(a_i, a_j) \leq f(a_j) - C(gw) < D(a_i, a_j)$;
- (3) WI if $d(a_i, a_j) \leq f(a_j) - C(gw) < M(a_i, a_j)$;
- (4) SI if $f(a_j) - C(gw) < d(a_i, a_j)$.

¹ Refer to [29, 34] for more details.

Definition 2. At run-time execution stage, at a_p ($p \leq i$), $F(a_i)$ is said to be of

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

For clarity, we further depict SC, WC, WI and SI in Figure 1.

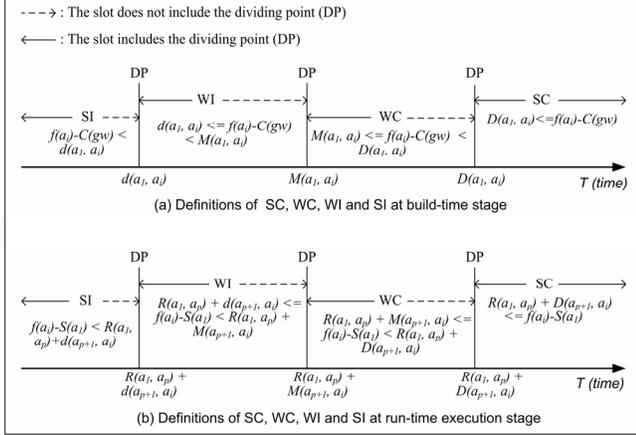


Figure 1. Definitions of SC, WC, WI and SI at build-time and run-time execution stages

According to [12], along grid workflow execution, for SC, we do not need to do anything as the corresponding fixed-time constraints can be kept. For WC, by utilising the possible time redundancy of the succeeding activity execution, the corresponding fixed-time constraints may still be kept. Specific methods for utilising the possible time redundancy can be found in [12]. For WI and SI, basically for most cases, the corresponding fixed-time 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 referred to [26].

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 fixed-time constraints to check their current consistency.

3. RELATED WORK AND PROBLEM ANALYSIS

3.1 Related Work

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

- CSS_1 : [19] takes every activity as a checkpoint. We denote this strategy as CSS_1 .
- CSS_2 : [41] sets checkpoints at the start time and end time of each activity. We denote this strategy as CSS_2 .
- CSS_3 : [32] 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 : [32] 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 : [10] selects activity a_i as a checkpoint if $R(a_i) > D(a_i)$. We denote this strategy as CSS_5 .
- CSS_6 : [11] selects activity a_i as a checkpoint if $R(a_i) > M(a_i)$. We denote this strategy as CSS_6 .
- CSS_7 : [13] 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 : [14] 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. To be uniform, we denote this strategy as CSS_8 although [14] denotes it as CSS_{MTR} (MTR : Minimum Time Redundancy).

3.2 Problem Analysis

All of $CSS_1 \sim CSS_7$ do not differentiate fixed-time constraints. Once an activity point is selected as a checkpoint, they will verify all fixed-time constraints. However, for some fixed-time constraints, their consistency can be deduced from others. Such constraints do not need to take any checkpoints and do not need to be verified. According to [14], CSS_8 can guarantee that at each selected checkpoint there is at least one fixed-time constraint violated. Since it treats all fixed-time constraints as a whole, it can claim that all checkpoints selected by it are “necessary” and “sufficient”. However, those fixed-time constraints whose consistency can be deduced from others do not need to take any checkpoints. That is to say, when we differentiate fixed-time constraints, CSS_8 has a similar problem of $CSS_1 \sim CSS_7$.

We now further illustrate the problem of $CSS_1 \sim CSS_8$. We consider three fixed-time constraints F_1 , F_2 and F_3 in a climate modelling grid workflow [2]. The grid workflow can contain hundreds of thousands of activities and sub activities such as discovering proper local climate models, data transfer, computing the impact of local thunderstorms on overall climate and so on [2]. Our example only relates to some of them, but is enough for analysing the problem of $CSS_1 \sim CSS_8$. We depict F_1 , F_2 and F_3 in Figure 2. We focus on SC of F_1 , F_2 and F_3 . The corresponding discussion for WC is similar.

In Figure 2, we suppose $f(F_1) = 9:00am$, $f(F_2) = 12:00pm$ and $f(F_3) = 6:00pm$. We consider an execution instance where $R(a_1) = 1$, $R(a_2) = 0.5$, $R(a_3) = 1.5$, $D(a_4) = 0.6$, $D(a_5) = 1$, $D(a_6) = 0.8$, $D(a_7) = 1.5$, $D(a_8) = 2$ and $D(a_9) = 1$. The time unit is hour. Suppose a_3 is selected as a checkpoint by one of $CSS_1 \sim CSS_8$. Then, at a_3 , all of F_1 , F_2 and F_3 will be verified according to Definition 2. We will find that F_1 is not of SC, but F_2 and F_3 are of SC. However, we argue that F_3 does not need to be verified because its consistency can be deduced from F_2 . That is to say, F_3 does not need to take a_3 as a checkpoint. We explain as follows. $R(a_1, a_3) + D(a_4, a_6) = R(a_1) + R(a_2) + R(a_3) + D(a_4) + D(a_5) + D(a_6) = 1 + 0.5 + 1.5 + 0.6 + 1 + 0.8 = 5.4$. We also have $f(F_2) - S(a_1) = 6$. Hence, we have inequation (1) below.

$$R(a_1, a_3) + D(a_4, a_6) < f(F_2) - S(a_1) \quad (1)$$

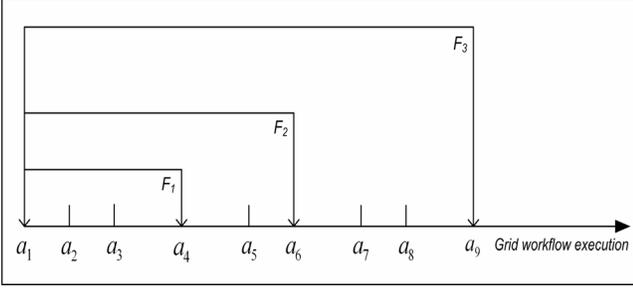


Figure 2. An example of three fixed-time constraints: F_1 , F_2 and F_3

According to Definition 2, inequation (1) means that F_2 is of SC. Meanwhile, we have $D(a_7, a_9) = D(a_7) + D(a_8) + D(a_9) = 1.5 + 2 + 1 = 4.5$. We also have $f(F_3) - f(F_2) = 6$. Hence, we have inequation (2) below.

$$D(a_7, a_9) < f(F_3) - f(F_2) \quad (2)$$

Combining inequations (1) and (2), we have $R(a_1, a_3) + D(a_4, a_9) = R(a_1, a_3) + D(a_4, a_6) + D(a_7, a_9) < f(F_2) - S(a_1) + f(F_3) - f(F_2) = f(F_3) - S(a_1)$. Accordingly, we have inequation (3) below.

$$R(a_1, a_3) + D(a_4, a_9) < f(F_3) - S(a_1) \quad (3)$$

According to Definition 2, inequation (3) means that F_3 is of SC.

The above example has demonstrated that we do not need to verify F_3 . We have actually deduced the consistency of F_3 from F_2 . Therefore, a_3 is a real checkpoint for F_1 and F_2 but not for F_3 . That is to say, we should differentiate fixed-time constraints and select checkpoints corresponding to different fixed-time constraints.

Considering the above example again, we can find that there is a key factor for us to deduce the consistency of F_3 from F_2 . That is $D(a_7, a_9) < f(F_3) - f(F_2)$. $D(a_7, a_9) < f(F_3) - f(F_2)$ is exactly the essence of temporal dependency between F_2 and F_3 . We detail it in the next section.

4. TEMPORAL DEPENDENCY

In this section, we discuss temporal dependency between fixed-time constraints. In general, temporal dependency means that different fixed-time 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 fixed-time constraint verification, temporal dependency consists of SC temporal dependency and WC temporal dependency. The former is for SC fixed-time constraints while the latter is for WC ones.

We first detail SC and WC temporal dependency in Section 4.1. Then in Section 4.2, we investigate how to deduce the consistency of fixed-time 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. More generally than in Figure 2, we consider N fixed-time constraints F_1, F_2, \dots, F_N in a climate modelling grid workflow [2]. We suppose that they are at $a_{j_1}, a_{j_2}, \dots, a_{j_N}$ respectively ($j_1 < j_2 < \dots < j_N$). As stated in Section 2, we focus on one execution path without losing generality. Correspondingly, we depict F_1, F_2, \dots, F_N in Figure 3.

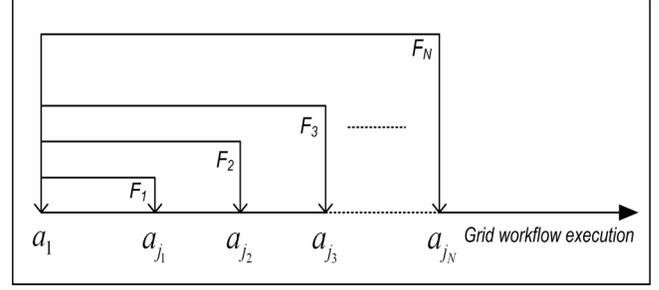


Figure 3. A series of fixed-time constraints

We first discuss SC temporal dependency between two fixed-time constraints and then extend it to multiple ones. We consider F_1 and F_2 in Figure 3. If $f(F_2) \leq f(F_1)$ due to an incorrect setting, then according to Definition 1, if F_2 is of SC, F_1 must also be of SC. If F_2 is not of SC, we need to adjust F_2 . Then even if F_1 is of SC, the adjustment inevitably influences F_1 because F_1 is included in F_2 . This will result in the necessity of re-verification of F_1 . That is to say, the previous verification of F_1 is ineffective. Therefore, in Figure 3, we must ensure $f(F_2) > f(F_1)$.

Now, we consider a more complicated case for F_1 and F_2 . At build-time stage, according to Definition 1, if F_1 and F_2 are of SC, we have inequations (4) and (5) below.

$$D(a_1, a_{j_1}) \leq f(F_1) - C(gw) \quad (4)$$

$$D(a_1, a_{j_2}) \leq f(F_2) - C(gw) \quad (5)$$

Then, if we omit the temporal dependency between them and set up F_1 and F_2 independently, we may encounter the following problem at run-time execution stage. At point a_{j_1} , to ensure F_2 is of SC, according to Definition 2, we must ensure that inequation (6) below holds.

$$R(a_1, a_{j_1}) + D(a_{j_1+1}, a_{j_2}) \leq f(F_2) - S(a_1) \quad (6)$$

We consider an extreme case where $E(a_{j_1}) = f(F_1)$. Then, F_1 can still be kept and we have $R(a_1, a_{j_1}) = f(F_1) - S(a_1)$. If we apply this equation to inequation (6), we can deduce that we must ensure inequation (7) below holds.

$$D(a_{j_1+1}, a_{j_2}) \leq f(F_2) - f(F_1) \quad (7)$$

The problem is that we cannot guarantee inequation (7) from inequations (4) and (5) that we only have. In fact, inequation (7) may or may not hold. If inequation (7) does not hold, then at a_{j_1} , F_2 is definitely not of SC even if the succeeding grid workflow execution after a_{j_1} has not started. This means that the setting of F_1 and/or F_2 at the build-time stage is not appropriate. Hence, we should adjust one or both of F_1 and F_2 . Consequently, we need to verify one or both of them again. Then, the previous temporal verification of one or both of them becomes ineffective. Therefore, we must consider the temporal dependency between F_1 and F_2 and ensure that inequation (7) always holds in order to keep the previous temporal verification effective. Correspondingly, we have Definition 3.

Definition 3 (SC Temporal Dependency). Let F_1 and F_2 be two fixed-time constraints at a_{j_1} and a_{j_2} respectively ($j_1 < j_2$) (see

Figure 3). Then, with $D(a_{j_1+1}, a_{j_2}) \leq f(F_2) - f(F_1)$, SC temporal dependency between F_1 and F_2 is defined as consistent.

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

Definition 4 (WC Temporal Dependency). Let F_1 and F_2 be two fixed-time constraints at a_{j_1} and a_{j_2} respectively ($j_1 < j_2$) (see Figure 3). Then, with $M(a_{j_1+1}, a_{j_2}) \leq f(F_2) - f(F_1)$, WC temporal dependency between F_1 and F_2 is defined as consistent.

Regarding SC and WC temporal dependency between F_1, F_2, \dots, F_N , we derive Theorem 1 below. With Theorem 1, SC or WC temporal dependency between F_1, F_2, \dots, F_N can be translated into that between two fixed-time constraints such as between F_1 and F_2 .

Theorem 1. Let F_1, F_2, \dots, F_N be N fixed-time constraints (see Figure 3) at $a_{j_1}, a_{j_2}, \dots, a_{j_N}$ respectively ($j_1 < j_2 < \dots < j_N$). Then,

- 1) if SC temporal dependency between any two adjacent fixed-time constraints F_k and F_{k+1} is consistent ($1 \leq k \leq N-1$), SC temporal dependency between any two non-adjacent fixed-time constraints must also be consistent;
- 2) if WC temporal dependency between any two adjacent fixed-time constraints F_k and F_{k+1} is consistent ($1 \leq k \leq N-1$), WC temporal dependency between any two non-adjacent fixed-time constraints must also be consistent.

Proof: 1) For simplicity, we consider F_1, F_2 and F_3 . Suppose SC temporal dependency between F_1 and F_2 is consistent and SC temporal dependency between F_2 and F_3 is consistent. Now we prove that SC temporal dependency between F_1 and F_3 is also consistent. That is to say, the consistency of temporal dependency is transitive. According to Definition 3, we have inequations (8) and (9) below.

$$f(F_1) + D(a_{j_1+1}, a_{j_2}) \leq f(F_2) \quad (8)$$

$$f(F_2) + D(a_{j_2+1}, a_{j_3}) \leq f(F_3) \quad (9)$$

Based on inequations (8) and (9), we have: $f(F_1) + D(a_{j_1+1}, a_{j_3}) = f(F_1) + D(a_{j_1+1}, a_{j_2}) + D(a_{j_2+1}, a_{j_3}) \leq f(F_2) + D(a_{j_2+1}, a_{j_3}) \leq f(F_3)$. Then, we have inequation (10) below.

$$f(F_1) + D(a_{j_1+1}, a_{j_3}) \leq f(F_3) \quad (10)$$

According to Definition 3, inequation (10) means that SC temporal dependency between F_1 and F_3 is consistent.

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

Thus, in overall terms, the theorem holds. ■

4.2 Consistency of Fixed-time Constraints

According to Section 4.1, at build-time stage we need to verify temporal dependency between any two adjacent fixed-time constraints and accordingly make sure of its consistency. After that, at run-time execution stage, we can derive Theorem 2 below. With Theorem 2, we can deduce the consistency of later fixed-time constraints from previous ones.

Theorem 2. Let F_1, F_2, \dots, F_N be N fixed-time constraints (see Figure 3) at $a_{j_1}, a_{j_2}, \dots, a_{j_N}$ respectively ($j_1 < j_2 < \dots < j_N$). Then,

with build-time SC and WC consistency of temporal dependency, at a_p between a_l and a_k ,

- 1) if F_k is of SC, any fixed-time constraint F_s after F_k must also be of SC ($k < s \leq N$);
- 2) if F_k is of WC, any fixed-time constraint F_s after F_k must also be of WC or even SC ($k < s \leq N$).

Proof: 1) If F_k is of SC, we have inequation (11) below.

$$R(a_l, a_p) + D(a_{p+1}, a_{j_k}) \leq f(F_k) - S(a_l) \quad (11)$$

Besides, with build-time verification of temporal dependency, we can make sure of SC consistency of temporal dependency between two adjacent fixed-time constraints. According to Theorem 1, temporal dependency between F_k and F_s must also be of SC. Thus, we have inequation (12) below.

$$f(F_k) + D(a_{j_k+1}, a_{j_s}) \leq f(F_s) \quad (12)$$

If we add (11) and (12) together, we have $R(a_l, a_p) + D(a_{p+1}, a_{j_k}) + f(F_k) + D(a_{j_k+1}, a_{j_s}) \leq f(F_k) - S(a_l) + f(F_s)$. Further, we have inequation (13) below.

$$R(a_l, a_p) + D(a_{p+1}, a_{j_s}) \leq f(F_s) - S(a_l) \quad (13)$$

According to item 1 of Definition 2, inequation (13) means that F_s is of SC.

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

Thus, in overall terms, the theorem holds. ■

5. CHECKPOINT SELECTION BASED ON SC AND WC TEMPORAL DEPENDENCY

Among $CSS_7 \sim CSS_8$, [14] 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. According to [14], CSS_8 can guarantee that at each checkpoint there is at least one fixed-time constraint violated. However, as analyzed in Section 3.2, it does not differentiate fixed-time constraints and will verify all of them at each checkpoint. In fact, a checkpoint is needed for some fixed-time constraints, but not needed for those whose consistency can be deduced without further verification. As analyzed in Section 4, with SC and WC temporal dependencies, we can derive such fixed-time 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_{TD} (TD: Temporal Dependency). In general, the working process of CSS_{TD} is as follows. We first apply CSS_8 to determine whether an activity point is selected as a checkpoint for all fixed-time constraints as a whole. Then, we apply SC and WC temporal dependencies to determine which fixed-time constraints should take the checkpoint for verification.

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

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 grid 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, it has been proved in [14] that there will be at least one SC or WC fixed-time constraint violated. Then, CSS_8 selects a_p as a checkpoint, but for verifying all fixed-time constraints.

5.1.1 Minimum SC and WC Time Redundancies

According to [14], at a_p , each SC fixed-time constraint has a SC time redundancy defined in Definition 5. Each WC fixed-time 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 ($p < i$), let $F(a_i)$ be of SC. Then, SC time redundancy of $F(a_i)$ at a_p is defined as $[f(a_i) - S(a_i)] - [R(a_i, a_p) + D(a_{p+1}, a_i)]$ and is denoted as $TR_{SC}(F(a_i), a_p)$ (TR : Time Redundancy).

Definition 6. At activity point a_p ($p < j$), let $F(a_j)$ be of WC. Then, the WC time redundancy of $F(a_j)$ at a_p is defined as $[f(a_j) - S(a_j)] - [R(a_i, a_p) + M(a_{p+1}, a_j)]$ and is denoted as $TR_{WC}(F(a_j), a_p)$.

Definition 7 (Minimum SC Time Redundancy). Let F_1, F_2, \dots, F_N be N SC fixed-time constraints and all of them cover activity point a_p . Then, at a_p , the minimum SC time redundancy is defined as the minimum of all SC time redundancies of F_1, F_2, \dots, F_N , and is denoted as $MTR_{SC}(a_p)$ (MTR : Minimum Time Redundancy).

$$MTR_{SC}(a_p) = \text{Min} \{ TR_{SC}(F_s, a_p) \mid s = 1, 2, \dots, N \}$$

Definition 8 (Minimum WC Time Redundancy). Let F_1, F_2, \dots, F_N be N WC fixed-time constraints and all of them cover activity point a_p . Then, at a_p , the minimum WC time redundancy is defined as the minimum of all WC time redundancies of F_1, F_2, \dots, F_N , and is denoted as $MTR_{WC}(a_p)$.

$$MTR_{WC}(a_p) = \text{Min} \{ TR_{WC}(F_s, a_p) \mid s = 1, 2, \dots, N \}$$

Obviously, 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 SC and WC Time Redundancies

[14] has 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 along grid workflow execution. In general, DOMTR works as follows. Along grid 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, $MTR_{SC}(a_p) = MTR_{SC}(a_{p-1}) - [Rcd(a_p) - D(a_p)]$ and $MTR_{WC}(a_p) = MTR_{WC}(a_{p-1}) - [Rcd(a_p) - M(a_p)]$. There is an exception for the computation. That is when a_p is the end activity of the fixed-time 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 fixed-time 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 minimum one of all

remaining SC time redundancies and the minimum one of all remaining WC time redundancies are initialised at run-time instantiation stage, i.e. before workflow execution. The detailed discussion can be found in [14].

5.1.3 Checkpoint Selection of CSS_8

[14] has concluded and demonstrated the relationships between minimum SC & WC time redundancies and SC, WC, WI & SI of fixed-time constraints. Based on those relationships, it proposed CSS_8 . We depict the relationships in Figure 4. In Figure 4, ‘‘previous’’ means before the execution of a_p .

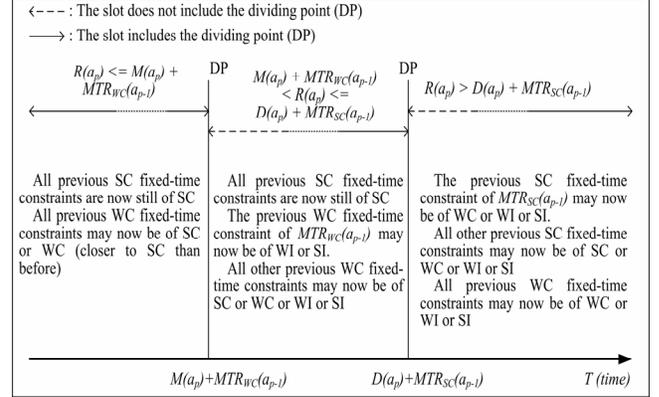


Figure 4. Relationships between minimum SC and WC time redundancies and SC, WC, WI & SI

According to Figure 4, at a_p , the following three conclusions are drawn in [14].

- 1) If $R(a_p) > D(a_p) + MTR_{SC}(a_{p-1})$, we need to verify all previous SC and WC fixed-time constraints. There can be multiple violations. At least, the fixed-time constraint whose SC time redundancy at a_{p-1} is $MTR_{SC}(a_{p-1})$ is violated and now is not of SC.
- 2) If $M(a_p) + MTR_{WC}(a_{p-1}) < R(a_p) \leq D(a_p) + MTR_{SC}(a_{p-1})$, we do not need to verify all previous SC fixed-time constraints, only all previous WC ones. There can be multiple violations. At least, the fixed-time constraint whose WC time redundancy at a_{p-1} is $MTR_{WC}(a_{p-1})$ is violated and now is not of SC and WC.
- 3) If $R(a_p) \leq M(a_p) + MTR_{WC}(a_{p-1})$, we do not need to verify all previous SC fixed-time constraints. As to previous WC ones, based on [12] we do not need to verify them either. In [12], a method has been developed to adjust the WC fixed-time constraints so that they can still be kept as SC. [14] has proved that after execution of a_p , the status of the previous WC fixed-time constraints is changed closer to SC (can even be changed to SC). Therefore, if a previous WC fixed-time 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. According to [14], 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 fixed-time constraints, and for verifying WC, WI & SI of all previous WC fixed-time 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 fixed-time 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. [14] has proved that with CSS_8 , at each checkpoint there is at least one fixed-time constraint violated.

5.2 Checkpoint Selection Process of CSS_{TD}

As stated earlier, CSS_{TD} applies CSS_8 to determine whether the current activity point is selected as a checkpoint for all fixed-time constraints as a whole. To be different from final real checkpoints, we call such a checkpoint as a tentative checkpoint. Then, CSS_{TD} applies temporal dependency to figure out which fixed-time 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_{TD} . The core part of it is depicted in Algorithm 1 below.

Input	Maximum, minimum and mean durations of all activities; ArraySC: an array of all SC fixed-time constraints; ArrayWC: an array of all WC fixed-time constraints.
Output	<i>True</i> or <i>False</i> as an appropriate checkpoint
Step 1	Compute $MTR_{SC}(a_p)$ and $MTR_{WC}(a_p)$ by DOMTR when grid workflow execution arrives at a_p .
	1.1. If a_p is not the end activity of the fixed-time constraint of $MTR_{SC}(a_{p-1})$, then $MTR_{SC}(a_p) = MTR_{SC}(a_{p-1}) - [Rcd(a_p) - D(a_p)]$ 1.2. If a_p is not the end activity of the fixed-time constraint of $MTR_{WC}(a_{p-1})$, then $MTR_{WC}(a_p) = MTR_{WC}(a_{p-1}) - [Rcd(a_p) - M(a_p)]$ 1.3. If a_p is the end activity of the fixed-time constraint of $MTR_{SC}(a_{p-1})$, then $MTR_{SC}(a_p) =$ minimum one of all remaining SC time redundancy; 1.4. If a_p is the end activity of the fixed-time constraint of $MTR_{WC}(a_{p-1})$, then $MTR_{WC}(a_p) =$ minimum one of all remaining WC time redundancy;
Step 2	Determine whether a_p is selected as a tentative checkpoint for all fixed-time 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 fixed-time 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 fixed-time 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 fixed-time constraints should take it as a real checkpoint and which ones should not.

3.1. At a_p , verify fixed-time constraints until a SC or WC one. Then, all fixed-time constraints after the SC or WC one do not take a_p as a real checkpoint while those before the SC or WC one as well as itself need to take a_p as a real checkpoint.

Algorithm 1. Checkpoint selection process of CSS_{TD}

6. COMPARISON AND SIMULATION

As analysed in Section 3, existing representative checkpoint selection strategies do not differentiate fixed-time constraints. Each checkpoint is for verifying all fixed-time constraints. This will cause some unnecessary temporal verification because we do not need to verify those fixed-time constraints whose consistency can be deduced without further verification. According to Section 5, CSS_{TD} uses temporal dependency to derive the consistency of later fixed-time constraints from previous fixed-time constraints. By this, CSS_{TD} can identify those fixed-time constraints whose consistency can be deduced without further verification. These fixed-time 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_{TD} , we can achieve better temporal verification efficiency.

We now perform an experimental simulation in our real-world grid workflow management system called SwinDeW-G (Swinburne Decentralised Workflow for Grid) [38, 39]. Our aim is to simulate temporal verification based on CSS_8 and CSS_{TD} at run-time execution stage to demonstrate that CSS_{TD} can improve temporal verification efficiency significantly than CSS_8 . [14] has experimentally demonstrated that CSS_8 can improve checkpoint selection and eventually temporal verification efficiency significantly than other existing representative strategies. Therefore, if our aim is achieved, we will be able to conclude that CSS_{TD} can improve temporal verification efficiency significantly over all existing representative strategies.

In Section 6.1, we describe the simulation environment. We then detail the simulation process in Section 6.2. In Section 6.3, we depict and analyse the simulation results to demonstrate the significant improvement of CSS_{TD} on temporal verification efficiency over CSS_8 .

6.1 Simulation Environment

The key component in our simulation environment is SwinDeW-G which is running on a grid testbed named SwinGrid (Swinburne Grid) [39]. An overall picture of SwinGrid is depicted in the bottom plane of Figure 5 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 CITR (Centre for Information Technology Research) Node, Swinburne ESR (Enterprise Systems Research laboratory) Node, Swinburne Astrophysics Supercomputer Node, and Beihang CROWN Node in China. They are running Linux, GT (Globus Toolkit) 4.03 or CROWN grid toolkits 2.5 [17, 39] where CROWN (China R&D Environment Over Wide-area Network) is an extension of GT4.03 with more middleware, hence compatible with GT4.03. 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 APAC (Australian Partnership for Advanced Computing) 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 grid workflow management system [39]. A grid 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 5, each grid node can have a number of peers. A peer can be simply viewed as a grid service [39]. In the top plane of Figure 5, we show a sample of how a grid workflow can be executed in the simulation environment.

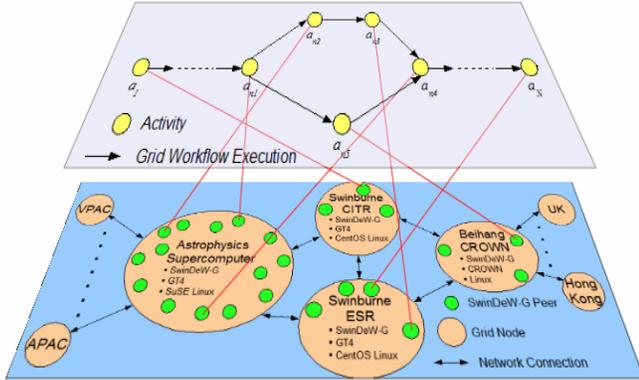


Figure 5. Overview of the Simulation Environment

6.2 Simulation Process

We have simulated temporal verification on a climate modelling grid workflow with CSS_{TD} and CSS_8 . Fixed-time constraints in it are as shown in Figure 3. 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 grid workflow execution, it executes CSS_8 to choose checkpoints. Then, at each checkpoint, it verifies all fixed-time constraints. After it finishes all checkpoints and all fixed-time 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_{TD} to choose appropriate checkpoints. Then, at each checkpoint, it verifies those fixed-time constraints which take the checkpoint as a real one. After it finishes all checkpoints and corresponding fixed-time constraints, we will have another number of all verification units. We denote it as $V(CSS_{TD})$.

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

6.3 Simulation Results and Analysis

Based on the simulation process described in Section 6.2, we can derive $V(CSS_{TD})$ and $V(CSS_8)$. They, together with corresponding

trajectories, are depicted in Figure 6. They change by the number of fixed-time constraints.

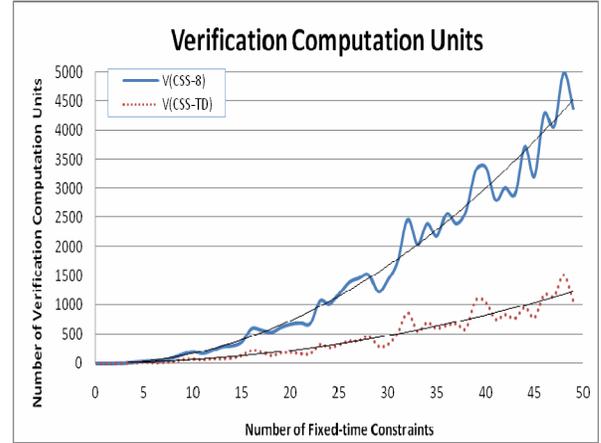


Figure 6. Verification Computation Units by CSS_{TD} and CSS_8

From Figure 6, we can see that with the number of fixed-time constraints increasing, both $V(CSS_{TD})$ and $V(CSS_8)$ increase. This is because with more fixed-time constraints, there will be more verification of them. Hence, there will be more $V(CSS_{TD})$ and $V(CSS_8)$. However, $V(CSS_8)$ goes up dramatically while $V(CSS_{TD})$ rises slowly. Especially, we can see that when the number of fixed-time constraints is getting larger, $V(CSS_8)$ is getting much greater than $V(CSS_{TD})$. That is, the larger number of fixed-time constraints, the more significant improvement on verification efficiency by CSS_{TD} over CSS_8 . Since a grid workflow normally contains hundreds of thousands of activities and lasts a long time, a large number of fixed-time constraints are often needed so that the corresponding grid workflow can be monitored at various activities in order to ensure overall temporal correctness [2, 7]. Therefore, we can conclude that with CSS_{TD} , we can improve temporal verification efficiency significantly over CSS_8 .

In addition, [14] 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_{TD} , we can improve temporal verification efficiency significantly over all existing representative strategies $CSS_1 \sim CSS_8$.

7. CONCLUSIONS AND FUTURE WORK

In grid workflow systems, to verify fixed-time constraints at runtime execution stage, checkpoints are often selected so that the verification is conducted only at such checkpoints rather than at all activity points. However, this is a complex issue. The problem of existing representative checkpoint selection strategies is that they do not differentiate fixed-time constraints as once a checkpoint is selected, it is for verifying all fixed-time constraints including those whose consistency can be deduced from others. Such fixed-time constraints actually do not need to take any checkpoints. Consequently, the verification of them is unnecessary, which can severely impact the efficiency of overall temporal verification since there are normally a large number of fixed-time constraints in a grid workflow. The efficiency of overall temporal verification is

important because it directly affects whether current temporal verification is useful.

To address the above problem, in this paper, a new checkpoint selection strategy named CSS_{TD} (Temporal Dependency based Checkpoint Selection Strategy) has been developed. CSS_{TD} can make checkpoint selection corresponding to different fixed-time constraints. Specifically, temporal dependency between fixed-time constraints has been identified and investigated intensively. With temporal dependency, the consistency of some later fixed-time constraints can be deduced from previous ones. Then, based on temporal dependency, CSS_{TD} was presented. With CSS_{TD} , those later fixed-time constraints whose consistency can be deduced from previous ones will no longer take any checkpoints. Accordingly, their verification can be avoided which otherwise incurred by existing representative strategies. The final comprehensive comparison and experimental simulation have shown that compared to existing representative strategies, CSS_{TD} can improve the efficiency of overall temporal verification significantly.

With these contributions, we can further investigate issues such as temporal exception handling when a fixed-time constraint is violated at a checkpoint. This could include dynamic negotiations between different grid services to compensate for the time deficit.

8. ACKNOWLEDGMENTS

The authors are grateful for the simulation work of R. Moore, the support from the CROWN team, and the English proofreading by G. Foster. The work reported in this paper is partly supported by Australian Research Council Projects under grant No. DP0663841 and grant No. LP0669660, by Swinburne Dean's Collaborative Grants Scheme 2007-2008, and by Swinburne Research Development Scheme 2008.

9. REFERENCES

- [1] 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 (2003), 5-51.
- [2] Abramson, A., 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 (2005), 69-78.
- [3] Al-Ali, R., Amin, K., Laszewski, G.V., Rana, O., Walker, D., Hategan, M., and Zaluzec, N. 2004. Analysis and Provision of QoS for Distributed Grid Applications. *Journal of Grid Computing*, 2, 2 (2004), 163-182.
- [4] Aloisio, G., Cafaro, M., Carteni, G., Epicoco, I., Quarta, G., Raolil, A. 2005. GridFlow for Earth Observation Data Processing. In *Proceedings of the 2005 International Conference on Grid Computing and Applications (Las Vegas, Nevada, USA, June 2005)*. GCA 2005. IEEE CS Press, 168-176.
- [5] Brandic, I., Pllana, S., Benkner, S. 2006. An Approach for the High-level Specification of QoS-aware Grid Workflows Considering Location Affinity. *Scientific Programming Journal*, 14, 3-4 (2006), 231-250.
- [6] Brandic, I., Pllana, S., Benkner, S. 2007. Specification, Planning, and Execution of QoS-aware Grid Workflows within the Amadeus Environment. *Concurrency and Computation: Practice and Experience*, 2007, in press.
- [7] Buyya, R., Abramson, D., and Venugopal, S. 2005. The Grid Economy. *Proceedings of The IEEE*, 93, 3 (2005), 698-714.
- [8] Cao, J., Jarvis, S. A., Saini, S., and Nudd, G. R. 2003. GridFlow: Workflow Management for Grid Computing. In *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (Tokyo, May 2003)*. CCGrid 2003. IEEE CS Press, 198-205.
- [9] Cybok, D. 2006. A Grid Workflow Infrastructure. *Concurrency and Computation: Practice and Experience, Special Issue on Workflow in Grid Systems*, 18, 10 (2006), 1243-1254.
- [10] 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)*. APSEC2004. IEEE CS Press, 30-37.
- [11] Chen, J., and Yang, Y. 2006. Activity Completion Duration based Checkpoint Selection for Dynamic Verification of Temporal Constraints in Grid Workflow Systems. *International Journal of High Performance Computing Applications*, Sage, 2006, in press.
- [12] Chen, J., and Yang, Y. 2007. Multiple States based Temporal Consistency for Dynamic Verification of Fixed-time Constraints in Grid Workflow Systems. *Concurrency and Computation: Practice and Experience*, 19, 7 (2007), 965-982.
- [13] Chen, J., and Yang, Y. 2005. 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)*. APSEC2005. IEEE CS Press, 299-306.
- [14] 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 (June 2007), Article 6.
- [15] Chen, L., Wassermann, B., Emmerich, W., and Foster, H. 2006. Web Service Orchestrations with BPEL. In *Proceedings of the 28th International Conference on Software Engineering (Shanghai, China, May 2006)*. ICSE2006. ACM Press, 2006, 1071-1072.
- [16] Chinn, S., and Madey, G. 2000. Temporal Representation and Reasoning for Workflow in Engineering Design Change Review. *IEEE Transactions on Engineering Management*, 47, 4 (2000), 485-492.
- [17] CROWN Team. 2007. CROWN portal, <http://www.crown.org.cn/en/>, accessed on Sept. 10, 2007.
- [18] Deelman, E., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., and Vahi, K. 2003. Mapping Abstract Complex Workflows onto Grid Environments. *Journal of Grid Computing*, 1, 1 (2003), 9-23.
- [19] 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). CAiSE'99. Springer Verlag, LNCS 1626, 286-300.
- [20] Emmerich, W., Butchart, B., Chen, L., Wassermann, B., and Price, S.L. 2005. Grid Service Orchestration using the Business Process Execution Language (BPEL). *Journal of Grid Computing*, 3, 3-4 (2005), 283-304.
- [21] Fahringer, T., Pllana, S., and Villazon, A. 2004. A-GWL: Abstract Grid Workflow Language. In *Proceedings of the 4th International Conference on Computational Science, Part III (Krakow, Poland, June 2004)*. ICCS2004. Springer Verlag, LNCS 3038, 42-49.
- [22] Foster, I., Kesselman, C., Nick, J., and Tuecke, S. 2002. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. In *Proceedings of the 5th Global Grid Forum Workshop (Edinburgh, Scotland, July 2002)*. GGF5.
- [23] Fox, G.C., and Gannon, D. 2006. Workflow in Grid Systems. *Concurrency and Computation: Practice and Experience*, 18, 10 (2006), 1009-1019.
- [24] Goble, C. 2004. Building ad hoc (personal) workflows in an open world: ^{my}Grid experiences. In *Proceedings of the 12th Global Grid Forum Workshop (Brussels, Belgium, Sept. 2004)*. GGF12.
- [25] Gomes, C., Rana, O., and Cunha, J. 2007. Extending Grid-based Workflow Tools with Patterns/Operators. *International Journal of High Performance Computing Applications*, 2007, in press.
- [26] Hagen, C., and Alonso, G. 2000. Exception Handling in Workflow Management Systems. *IEEE Transactions on Software Engineering*, 26, 10 (2000), 943-958.
- [27] Laszewski, G.V., Amin, K., Hategan, M., Zaluzec, N.J., Hampton, S., and Rossi, A. 2004. GridAnt: A Client-Controllable Grid Workflow System. In *Proceedings of the 37th Hawaii International Conference on System Sciences (Hawaii, USA, January 2004)*. HICSS'04. 210-219.
- [28] Li, H., Yang, Y., and Chen, T.Y. 2004. Resource Constraints Analysis of Workflow Specifications. *The Journal of Systems and Software*, 73, 2 (2004), 271-285.
- [29] 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 (2003), 179-193.
- [30] 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 (2006), 1039-1065.
- [31] Marinescu, D. 2002. A Grid Workflow Management Architecture. *Global Grid Forum White Paper*, Aug. 2002, http://www-unix.gridforum.org/mail_archive/gcewg/2002/Archive/pdf00003.pdf, accessed on Sept. 10, 2007.
- [32] Marjanovic, O., and Orłowska, M. E. 1999. On Modeling and Verification of Temporal Constraints in Production Workflows. *Knowledge and Information Systems*, 1, 2 (1999), 157-192.
- [33] 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 (2006), 1067-1100.
- [34] Sadiq, W., and Orłowska, M.E. 2000. Analysing Process Models using Graph Reduction Techniques. *Information Systems*, 25, 2 (2000), 117-134.
- [35] Simpson, D. R., Kelly, N., Jithesh, P. V., Donachy, P., Harmer, T., J., Perrott, R. H., Johnston, J., Kerr, P., McCurley, M., and McKee, S. 2004. GeneGrid: A Practical Workflow Implementation for a Grid Based Virtual Bioinformatics Laboratory. In *Proceedings of the UK e-Science All Hands Meeting 2004 (UK, Sept. 2004)*. AHM2004. 547-554.
- [36] Son, J. H., and Kim, M. H. 2001. Improving the Performance of Time-constrained Workflow Processing. *The Journal of Systems and Software*, 58, 3 (2001), 211-219.
- [37] Taylor, I., Shields, M., Wang, I., and Harrison, A. 2005. Visual Grid Workflow in Triana. *Journal of Grid Computing*, 3, 3-4 (2005), 153-169.
- [38] 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 (2006), 922-935.
- [39] Yang, Y., Liu, K., Chen, J., Lignier, J., and Jin, H. 2007. Peer-to-Peer Based Grid Workflow Runtime Environment of SwinDeW-G. In *Proceedings of the 3rd International Conference on e-Science and Grid Computing (Bangalore, India, Dec. 2007)*. e-Science07. 51-58.
- [40] Yu, J., and Buyya, R. 2005. A Taxonomy of Workflow Management Systems for Grid Computing. *Journal of Grid Computing*, 3, 3 (2005), 171-200.
- [41] Zhuge, H., Cheung, T., and Pung, H. 2001. A Timed Workflow Process Model. *The Journal of Systems and Software*, 55, 3 (2001), 231-243.