A Minimum Proportional Time Redundancy based Checkpoint Selection Strategy 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
{jchen, yyang}@ict.swin.edu.au

Abstract

In grid workflow systems, existing typical checkpoint selection strategies, which are used to select checkpoints for verifying fixed-time constraints at run-time execution stage, are not effective and/or efficient for fixed-time constraint verification because they often ignore some necessary checkpoints and select some unnecessary checkpoints. To improve such status, in this paper, we develop a new checkpoint selection strategy. Specifically, we first address a new concept of minimum proportional time redundancy which can be used to tolerate certain time deviation incurred by abnormal grid workflow execution. Then, we discuss relationships between minimum proportional time redundancy and fixed-time constraint consistency. Based on the relationships, we present our new strategy. With the strategy, we can avoid the omission of necessary checkpoints and the selection of excess unnecessary checkpoints. Consequently, our strategy is more effective and efficient for fixed-time constraint verification than the existing typical strategies. The final evaluation further demonstrates this result.

1. Introduction

In the grid architecture, a grid workflow system is a type of high-level grid middleware which is supposed to support modelling, redesign and execution of large-scale sophisticated e-science and e-business processes in many complex scientific and business applications such as climate modelling, astrophysics, international finance and insurance [1, 3, 4, 10, 13]. Generally speaking, the whole working process of a grid workflow system can be divided into three stages: build-time, run-time instantiation and run-time execution [4, 15, 16]. At the build-time stage, complex scientific or business processes are modelled or redesigned as grid workflow specifications by some grid workflow definition languages such as Grid Services Flow Language (GSFL), Abstract Grid Workflow Language (AGWL), or Grid Workflow Execution Language (GWEL) [4, 12, 16]. Conceptually, a grid workflow contains a lot of computation, data or transaction intensive activities and dependencies between them [1, 15, 16]. These activities are implemented and executed by corresponding grid services [4, 16]. The dependencies define activity execution orders and form four basic control structures: sequential, parallel, selective and iterative [4, 16]. At the run-time instantiation stage, grid workflow instances are created, and especially grid services specified in build-time definition documents are discovered by an instantiation grid service [4, 16]. At the run-time execution stage, the grid workflow instances are executed, and the execution is coordinated between grid services by the grid workflow engine that is a high-level grid service [4, 16].

To control temporal correctness of grid workflow specification and execution, fixed-time constraints are often set at build-time [11, 18]. A fixed-time constraint at an activity is an absolute time value by which the activity must be completed. For example, a climate modelling grid workflow must be completed by the scheduled time [1], say 8:00pm, so that weather forecasting can be broadcast on time later. Here, 8pm is a fixed-time constraint. After fixed-time constraints are set, temporal verification is conducted to check whether they are all consistent. At the build-time and run-time instantiation stages, temporal verification is static because there are no any specific execution times. For each fixed-time constraint, we conduct its verification once only with the consideration of all covered activities. Therefore, we need not decide at which activities we should conduct the verification. At the run-time execution stage, activity completion duration is uncertain. Hence, we may need to verify a fixed-time constraint many times at different activities. However, conducting the verification at every activity is not efficient because we may not have to do so at some activities such as those which can be completed within allowed time intervals. So where should we conduct the temporal verification? The activities at which we conduct the verification are called checkpoints [5, 11, 18, 19]. This is the subject of the research field on CSS (Checkpoint Selection Strategies) [5, 11, 18, 19].
Some typical checkpoint selection strategies have been proposed. However, they often ignore some necessary checkpoints and select some unnecessary ones. Consequently, they often omit a lot of necessary temporal verification and incur some unnecessary temporal verification, which eventually will impact overall temporal verification effectiveness and efficiency. To improve such status, in this paper, we develop a new checkpoint selection strategy. Specifically, in Section 2, we detail related work and problem analysis for checkpoint selection. In Section 3, we describe a timed grid workflow representation. And in Section 4, we introduce a new concept of minimum proportional time redundancy and discuss how to obtain it. Then, In Section 5, we analyse relationships between minimum proportional time redundancy and temporal consistency. Based on the relationships, we present our new strategy. In Section 6, we further evaluate our strategy by quantitatively comparing it with existing typical strategies. The quantitative evaluation shows that our strategy can achieve better verification effectiveness and efficiency than the existing typical strategies. Finally in Section 7, we conclude our contributions and point out future work.

2. Related work and problem analysis for checkpoint selection

Different typical checkpoint selection strategies have been proposed in the literature. [11] takes every activity as a checkpoint. We denote this strategy as CSS1. [19] sets checkpoints at the start time and end time of each activity. We denote this strategy as CSS2. [18] takes the start activity as a checkpoint and adds a checkpoint after each decision activity is executed. We denote this strategy as CSS3. [18] also mentions another checkpoint selection strategy: user-defined static checkpoints such as user-prescribed static time points. We denote this strategy as CSS4. All of CSS1, CSS2, CSS3 and CSS4 predefine checkpoints before grid workflow execution. However, since activity completion durations vary, we may not have to conduct temporal verification at some of these predefined checkpoints such as those that can be completed within allowed time intervals. Therefore, CSS1, CSS2, CSS3 and CSS4 may select some unnecessary checkpoints. Hence, CSS1, CSS2, CSS3 and CSS4 are not efficient for temporal verification. In addition, although CSS2 and CSS3 do not ignore any checkpoints at the heavy cost of inefficiency, CSS3 and CSS4 may ignore some checkpoints as we may need to conduct temporal verification at some other activities rather than the checkpoints predefined by CSS2 or CSS3. Therefore, CSS2 and CSS3 are not effective for temporal verification.

Our earlier works [5, 7] have attempted to improve this situation, but they still have some deficiencies. Specifically, [5] introduces a maximum duration for each activity and then selects an activity as a checkpoint when its completion duration exceeds its maximum duration. We denote this strategy as CSS5. [7] introduces a mean duration for each activity and then, selects an activity as a checkpoint when its completion duration exceeds its mean duration. We denote this strategy as CSS6. However, in Section 6, we will see that under some conditions, we still need to select an activity as a checkpoint even if the above selection condition of CSS5 is not met, and under some other conditions, we need not select an activity as a checkpoint even if the above selection condition of CSS5 is met. In other words, CSS5 may ignore some necessary checkpoints while CSS6 may select some unnecessary checkpoints. Hence, CSS5 is not effective enough and CSS6 is not efficient enough for temporal verification.

Regarding the above limitations of the existing typical checkpoint selection strategies, in this paper, we develop a new checkpoint selection strategy which will be more effective and efficient for fixed-time constraint verification than the existing typical strategies.

3. Timed grid workflow representation

According to [4, 6, 9, 11], based on the directed graph concept, a grid workflow can be represented by a grid workflow graph, where nodes correspond to activities and edges correspond to dependencies between activities [9, 11]. We borrow some concepts from [11, 18] such as maximum, mean or minimum duration as a basis to represent grid workflow time attributes. We denote a grid workflow as $gw$, and the $i^{th}$ activity of $gw$ 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 $Rcd(a_i)$ 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 [6, 17], $D(a_i)$, $M(a_i)$ and $d(a_i)$ can be obtained by applying some stochastic models such as Poisson distribution or exponential distribution based on the past execution history. The past execution history covers delay time incurred at $a_i$ during the past execution such as setup delay, queuing delay, synchronisation delay, network latency and so on. The detailed discussion on how to obtain and set $D(a_i)$, $M(a_i)$ and $d(a_i)$ is outside the scope of this paper and can be found in [6, 11, 18]. For a specific execution of $a_i$, the delay time is included in $Rcd(a_i)$. Normally, we have $d(a_i) \leq M(a_i) \leq D(a_i)$ and $d(a_i) \leq Rcd(a_i) \leq D(a_i)$. If there is a fixed-time constraint at $a_i$, we denote it as $FTC(a_i)$ and its value as $ftv(a_i)$. If there is a path from $a_i$ to $a_j$ ($i \neq j$), we denote 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 $Rcd(a_i, a_j)$ respectively. Again, normally we have $d(a_i, a_j) \leq M(a_i, a_j) \leq D(a_i, a_j)$. 


Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC'05)

0-7695-2465-6/05 $20.00 © 2005 IEEE

Authorized licensed use limited to: SWINBURNE UNIV OF TECHNOLOGY. Downloaded on November 5, 2009 at 23:12 from IEEE Xplore. Restrictions apply.
For convenience, we consider one execution path in a grid workflow without losing generality. As to a selective or parallel structure, for each branch, it is an execution path. For an iterative structure, from start to end, it is also an execution path. Therefore, for the selective/parallel/iterative structure, we can also apply the results achieved from one execution path. Correspondingly, between $a_i$ and $a_{p}$, $D(a_i, a_{p})$ is equal to the sum of activity maximum durations, and $d(a_i, a_{p})$ is equal to the sum of activity minimum durations, and $M(a_i, a_{p})$ is equal to the sum of activity mean durations.

Besides the above time attributes, [6] defines four consistency states: SC (Strong Consistency), WC (Weak Consistency), WI (Weak Inconsistency) and SI (Strong Inconsistency). Because our new checkpoint selection strategy is intended to verify them, we summarise their definitions here. However, since the checkpoint concept is related to the run-time execution stage only and the minimum proportional time redundancy concept addressed in Section 4 is related to the run-time instantiation stage, we summarise the definitions for the two stages only. The definitions for the build-time stage and other detailed discussion can be found in [6].

**Definition 1.** At run-time instantiation stage, with $D(a_i, a_{p}) \leq ftv(a_i) - S(a_i)$, FTC$(a_i)$ is said to be of SC; with $M(a_i, a_{p}) \leq ftv(a_i) - S(a_i) < D(a_i, a_{p})$, FTC$(a_i)$ is said to be of WC; with $d(a_i, a_{p}) \leq ftv(a_i) - S(a_i) < M(a_i, a_{p})$, FTC$(a_i)$ is said to be of WI; and with $ftv(a_i) - S(a_i) < d(a_i, a_{p})$, FTC$(a_i)$ is said to be of SI.

**Definition 2.** At run-time execution stage, at checkpoint $a_i$ which is either before or at $a_p$, with $Rcd(a_i, a_{p}) + D(a_{p+1}, a_i) \leq ftv(a_i) - S(a_i)$, FTC$(a_i)$ is said to be of SC; with $Rcd(a_i, a_{p}) + M(a_{p+1}, a_i) \leq ftv(a_i) - S(a_i) < Rcd(a_i, a_{p}) + D(a_{p+1}, a_i)$, FTC$(a_i)$ is said to be of WC; with $d(a_i, a_{p}) + d(a_{p+1}, a_i) \leq ftv(a_i) - S(a_i) < Rcd(a_i, a_{p}) + M(a_{p+1}, a_i)$, FTC$(a_i)$ is said to be of WI; and with $ftv(a_i) - S(a_i) < Rcd(a_i, a_{p}) + d(a_{p+1}, a_i)$, FTC$(a_i)$ is said to be of SI.

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

According to [6], along grid workflow execution, for SC, we need not do anything as corresponding fixed-time constraints can be kept. For WC, by using possible time redundancy of succeeding activity execution, corresponding fixed-time constraints may still be kept. The specific methods on how to utilise the possible time redundancy can be found in [6]. For WI and SI, basically for most cases, corresponding fixed-time constraints cannot be kept. Consequently, exception handling will be triggered to adjust them to SC or WC. The specific exception handling discussion can be referred to [14].

Since WI and SI are adjusted to SC or WC by their respective exception handling, along grid workflow execution, the checkpoint selection is actually focused on selecting checkpoints for verifying previous SC and WC fixed-time constraints to check their current consistency.

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

4. Minimum proportional time redundancy

According to Section 3, since the checkpoint selection is actually focused on SC and WC fixed-time constraints, minimum proportional time redundancy consists of minimum SC proportional time redundancy and minimum WC proportional time redundancy. The former is for SC fixed-time constraints and the latter is for WC ones.

We now first introduce the concept of SC proportional time redundancy and WC proportional time redundancy from one fixed-time constraint in Section 4.1. Then, in Section 4.2, we introduce minimum SC and WC proportional time redundancy from multiple ones.

4.1. SC and WC proportional time redundancy

At the run-time instantiation stage, we consider one fixed-time constraint, say FTC$(a_i)$. If FTC$(a_i)$ is of SC, then, according to Definition 1, we have $D(a_i, a_{p}) \leq ftv(a_i) - S(a_i)$. Obviously, there is a time redundancy: $[ftv(a_i) - S(a_i) - D(a_i, a_{p})]$. We can allocate it to all activities covered by FTC$(a_i)$ according to certain allocating methods. Then, each activity will hold a time quota. The time quota is very useful for the grid workflow execution because it can be used to tolerate certain time deviation incurred by the abnormal grid workflow execution. We denote the time quota of FTC$(a_i)$ at $a_p$ as $TQ_{SC}(FTC(a_i), a_{p})$ and propose an allocating method next.

We first sort all $D(a_i) - M(a_i)$ $(j = 1, 2, 3, \ldots, i)$ in ascending order and we will get a sorting list. We denote the list as $L$ and the items in $L$ as $L_1, L_2, \ldots, L_i$. If $D(a_{p}) - M(a_{p})$ $(p \leq i)$ is ranked number $k$ in $L$, namely $L_k$, then we propose (1) below to allocate $TQ_{SC}(FTC(a_i), a_{p})$.
defined as the time quota allocated to
it as

\[ T_{QWC}(FTC(ai), ap) = \frac{[fiv(ai) - S(ai)] - D(ai, ap)}{i[\sum_{i=1}^{j} D(ai) - M(ai)]} \]

(1 ≤ k ≤ i, 1 ≤ p ≤ i) (1)

The relationship between \( L_k \) and \( L_{i,k+1} \) is depicted in Figure 2.

\[ \text{Figure 2. Relationship between } L_k \text{ and } L_{i,k+1} \]

In (1), we allocate \([fiv(ai) - S(ai)] - D(ai, ap)\) to activities covered by \( FTC(ai) \) based on the difference between the activity maximum duration and the activity mean duration. The activity with a bigger difference will be allocated a smaller quota of \([fiv(ai) - S(ai)] - D(ai, ap)\). This is because statistically, an activity can be completed around its mean duration. Therefore, the activity with a bigger difference between its maximum duration and its mean duration has more time to compensate possible time deviation incurred by abnormal grid workflow execution. Hence, we should allocate a smaller quota to it.

We define \( T_{QSC}(FTC(ai), ap) \) obtained by (1) as SC proportional time redundancy of \( FTC(ai) \) at \( ap \), and denote it as \( PTR_{SC}(FTC(ai), ap) \) (PTR: Proportional Time Redundancy). Correspondingly, we have Definition 3.

**Definition 3.** At activity \( ap \) \((p ≤ i)\), let \( FTC(ai) \) be of SC. Then, SC proportional time redundancy of \( FTC(ai) \) at \( ap \) is defined as the time quota allocated to \( ap \) by applying formula (1), and is denoted as \( PTR_{SC}(FTC(ai), ap) \).

If \( FTC(ai) \) is of WC, then, according to Definition 1, we have \( M(ai, ap) ≤ fiv(ai) - S(ai) < D(ai, ap) \). There is also a time redundancy which is \([fiv(ai) - S(ai)] - M(ai, ap)\). Similar to the above situation where \( FTC(ai) \) is of SC, we can allocate the time redundancy to all activities covered by \( FTC(ai) \) and then each activity will hold another time quota. We denote the time quota of \( FTC(ai) \) at \( ap \) as \( T_{QWC}(FTC(ai), ap) \). The corresponding allocating method is similar to (1). However, for clarity and convenience of the discussion, we depict it in (2).

\[ T_{QWC}(FTC(ai), ap) = \frac{[fiv(ai) - S(ai)] - M(ai, ap)}{i[\sum_{i=1}^{j} D(ai) - M(ai)]} \]

(1 ≤ k ≤ i, 1 ≤ p ≤ i) (2)

We define \( T_{QWC}(FTC(ai), ap) \) obtained by (2) as WC proportional time redundancy in Definition 4.

**Definition 4.** At activity \( ap \) \((p ≤ i)\), let \( FTC(ai) \) be of WC. Then, WC proportional time redundancy of \( FTC(ai) \) at \( ap \) is defined as the time quota allocated to \( ap \) by applying formula (2), and is denoted as \( PTR\_{WC}(FTC(ai), ap) \).

4.2. Minimum SC and WC proportional time redundancy

We firstly consider \( M \) SC fixed-time constraints \( F1, F2, \ldots, FM \) which all cover \( ap \). Then, according to Definition 3, each of \( F1, F2, \ldots, FM \) has a SC proportional time redundancy at \( ap \). We define minimum SC proportional time redundancy at \( ap \) as the minimum one of them, and denote it as \( MPTR_{SC}(ap) \) (MPTR: Minimum Proportional Time Redundancy). Then, we have Definition 5 below.

**Definition 5 (Minimum SC Proportional Time Redundancy).** Let \( F1, F2, \ldots, FM \) be \( M \) SC fixed-time constraints. All of \( F1, F2, \ldots, FM \) cover activity point \( ap \). Then, at \( ap \), the minimum SC proportional time redundancy is defined as the minimum one of all SC proportional time redundancies of \( F1, F2, \ldots, FM \), and is denoted as \( MPTR_{SC}(ap) \).

\[ MPTR_{SC}(ap) = \min\{ PTR_{SC}(Fk, ap) | k = 1, 2, \ldots, M \} \]

We now consider the situation where \( F1, F2, \ldots, FM \) are of WC. Based on Definition 4, we define the minimum WC proportional time redundancy in Definition 6.

**Definition 6 (Minimum WC Proportional Time Redundancy).** Let \( F1, F2, \ldots, FM \) be \( M \) WC fixed-time constraints. All of \( F1, F2, \ldots, FM \) cover activity point \( ap \). Then, at \( ap \), the minimum WC proportional time redundancy is defined as the minimum one of all WC proportional time redundancies of \( F1, F2, \ldots, FM \), and is denoted as \( MPTR_{WC}(ap) \).

\[ MPTR_{WC}(ap) = \min\{ PTR_{WC}(Fk, ap) | k = 1, 2, \ldots, M \} \]

Obviously, at \( ap \), just before the execution of \( ap \), the minimum SC and WC proportional time redundancy are \( MPTR_{SC}(ap) \) and \( MPTR_{WC}(ap) \) respectively.

In addition, we normally have \( M(ap) + MPTR_{WC}(ap) < D(ap) + MPTR_{SC}(ap) \). The reason is simple. If \( D(ap) + MPTR_{SC}(ap) \leq M(ap) + MPTR_{WC}(ap) \), then, since the fixed-time constraint of \( MPTR_{SC}(ap) \) is of SC, the one of \( MPTR_{WC}(ap) \) must also be of SC. But actually, the fixed-time constraint of \( MPTR_{WC}(ap) \) is of WC.

5. Checkpoint selection based on minimum SC and WC proportional time redundancy

5.1. Relationships between minimum SC & WC proportional time redundancy and SC, WC, WI & SI

At run-time execution stage, at \( ap \), we discuss relationships between \( MPTR_{SC}(ap) \) & \( MPTR_{WC}(ap) \) and SC, WC, WI & SI by deriving some theorems that will form the basis for presenting our checkpoint selection strategy. We will also address the run-time completion duration of \( ap \), namely \( Rcd(ap) \).

**Theorem 1.** At activity point \( ap \) if \( D(ap) + MPTR_{SC}(ap) < Rcd(ap) \), then, all previous WC fixed-time constraints 1)
now cannot be of SC and 2) may now be of WC, WI or SI; and 3) previous SC fixed-time constraints may now be of SC, WC, WI or SI.

**Proof:** 1) Suppose $FTC(a_p)$ is of WC before execution of $a_p (p\leq n)$, then, $ftv(a_p) - S(a_p) < Rcd(a_p, a_{p+1}) + D(a_{p+1}, a_m)$. If $D(a_p) + MTRSC(a_{p+1}) < Rcd(a_p)$, then, $ftv(a_p) - S(a_p) < Rcd(a_p, a_{p+1}) + D(a_p, a_m) = Rcd(a_p, a_{p+1}) + D(a_{p+1}, a_m)$. Meanwhile, we also have:

$$\text{fv}(a_p) - S(a_p) > Rcd(a_p, a_{p+1}) + D(a_{p+1}, a_m)$$

(3)

Obviously, according to Definition 2, (3) means that $FTC(a_p)$ cannot be of SC after the execution of $a_p$.

2) If $D(a_p) + MTRSC(a_{p+1}) < Rcd(a_p)$, then $Rcd(a_p, a_{p+1}) + M(a_{p+1}, a_m) > Rcd(a_p, a_{p+1}) + M(a_{p+1}, a_m) + M(a_{p+1}, a_m)$. Similarly, we may or may not have:

$$Rcd(a_p) + M(a_p, a_m) < Rcd(a_p) + M(a_p, a_m)$$

(4)

Similarly, we may or may not have $Rcd(a_p) + M(a_p, a_m)$.

3) Suppose $FTC(a_p)$ is of SC before execution of $a_p (p\leq n)$, according to Definition 2, we have:

$$Rcd(a_p, a_{p+1}) + D(a_{p+1}, a_m) \leq ftv(a_{p+1}) - S(a_p)$$

(5)

However, from (4) and (5), we cannot ensure (6) below.

$$Rcd(a_p, a_{p+1}) + M(a_{p+1}, a_m) \leq ftv(a_{p+1}) - S(a_p)$$

(6)

If (6) holds, $FTC(a_p)$ is of WC again. However, depending on specific $MTRWC(a_{p+1})$, (6) may or may not hold. Similarly, we may or may not have $Rcd(a_p, a_{p+1})$. If $rtv(a_p, a_{p+1})$, we may or may not have $Rcd(a_p, a_{p+1})$. However, from (7) and (8), we can not ensure (9) below.

$$Rcd(a_p, a_{p+1}) + D(a_{p+1}, a_m) \leq ftv(a_{p+1}) - S(a_p)$$

(9)

If (9) holds, $FTC(a_p)$ is of SC again. However, depending on specific $MTRWC(a_{p+1})$, (9) may or may not hold.

Theorem 2. At activity point $a_p$, if $D(a_p) + MTRSC(a_{p+1}) < Rcd(a_p)$, then $FTC(a_p)$ is of SC. Finally, we have:

$$Rcd(a_p, a_{p+1}) + D(a_{p+1}, a_m) \leq ftv(a_{p+1}) - S(a_p)$$

(10)

According to Definition 2, (10) means that $FTC(a_p)$ is still of SC after the execution of $a_p$.

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

Thus, the theorem holds.

**Theorem 3.** At activity point $a_p$, if $Rcd(a_p) \leq M(a_p) + MTRWC(a_{p+1})$, 1) all previous SC fixed-time constraints are now still of SC; and 2) all previous WC fixed-time constraints are now of either WC or SC; and 3) if they are still of WC, the status of them is closer to SC than before.

**Proof:** 1) The proof is similar to 1) of Theorem 2, hence omitted.

2) Suppose $FTC(a_p)$ is of WC before the execution of $a_p (p\leq n)$, according to Definition 2, we have:

$$Rcd(a_p, a_{p+1}) + D(a_{p+1}, a_m) \leq ftv(a_{p+1}) - S(a_p)$$

(11)

In addition, according to Definition 6, $MTRWC(a_{p+1}) < PTRWC(FTC(a_p), a_{p+1})$. Meanwhile, according to formula (2), $PTRWC(FTC(a_p), a_{p+1}) \leq ftv(a_{p+1}) - S(a_p) - MTRWC(a_{p+1})$. Therefore, we have:

$$Rcd(a_p, a_{p+1}) + D(a_{p+1}, a_m) \leq ftv(a_{p+1}) - S(a_p) - MTRWC(a_{p+1})$$

(12)

However, from (11) and (12) and (13) that we only have, we cannot judge whether (14) or (15) holds.

$$ftv(a_{p+1}) - S(a_p) < Rcd(a_p, a_{p+1}) + D(a_{p+1}, a_m)$$

(14)

In fact, depending on specific $MTRWC(a_{p+1})$, either of (14) and (15) may hold. If (14) holds, then, with (12), we have:

$$Rcd(a_p, a_{p+1}) + D(a_{p+1}, a_m) \leq ftv(a_{p+1}) - S(a_p) - MTRWC(a_{p+1})$$

(15)

According to Definition 2, (16) means that $FTC(a_p)$ is of SC. If (15) holds, according to Definition 2, $FTC(a_p)$ is already switched to be of SC after the execution of $a_p$.

In summary, $FTC(a_p)$ is of either SC or WC after execution of $a_p$.
Proceedings of the 12th Asia-Pacific Software Engineering Conference (APSEC'05)

Based on the above conclusions, we can decide whether we should take \( a_p \) as a checkpoint. Correspondingly we derive a new checkpoint selection strategy and denote it as CSS\textsubscript{MPTR} (Minimum Proportional Time Redundancy based Checkpoint Selection Strategy).

CSS\textsubscript{MPTR} is: At activity \( a_p \), if \( D(a_p) + MPTR\textsubscript{SC}(a_p, a_p) < Rcd(a_p) \), we take \( a_p \) 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 ones. If \( M(a_p) + MPTR\textsubscript{WC}(a_p, a_p) < Rcd(a_p) \leq D(a_p) + MPTR\textsubscript{SC}(a_p, a_p) \), we take \( a_p \) as a checkpoint for verifying SC, WC, WI & SI of all previous WC fixed-time constraints. If \( Rcd(a_p) \leq M(a_p) + MPTR\textsubscript{WC}(a_p, a_p) \), we do not take \( a_p \) as a checkpoint.

6. Comparison and quantitative evaluation

6.1. Overall comparison

According to Section 1, CSS\textsubscript{1} takes every activity as a checkpoint. CSS\textsubscript{2} sets checkpoints at the start time and end time of each activity. However, according to CSS\textsubscript{MPTR}, at an activity, say \( a_p \), if \( Rcd(a_p) \leq M(a_p) + MPTR\textsubscript{WC}(a_p, a_p) \), we need not take the activity as a checkpoint. Therefore, compared to CSS\textsubscript{MPTR}, CSS\textsubscript{2} and CSS\textsubscript{5} may select some unnecessary checkpoints. Hence, CSS\textsubscript{MPTR} is more efficient for temporal verification than CSS\textsubscript{1} and CSS\textsubscript{2}.

Similarly, we can derive that CSS\textsubscript{MPTR} is more efficient than CSS\textsubscript{1} and CSS\textsubscript{2}. In addition, at an activity, say \( a_p \), which is not defined as a checkpoint by CSS\textsubscript{1} and CSS\textsubscript{2}, if \( M(a_p) + MPTR\textsubscript{WC}(a_p, a_p) < Rcd(a_p) \), then according to CSS\textsubscript{MPTR}, we should take it as a checkpoint. In other words, CSS\textsubscript{3} and CSS\textsubscript{4} may ignore some necessary checkpoints. Hence CSS\textsubscript{MPTR} is also more effective than CSS\textsubscript{3} and CSS\textsubscript{4}.

According to the discussion in Section 5 for CSS\textsubscript{MPTR}, at activity \( a_p \), if \( M(a_p) + MPTR\textsubscript{WC}(a_p, a_p) < Rcd(a_p) \), we should take it as a checkpoint for verifying all previous WC fixed-time constraints because they may be violated. However, by CSS\textsubscript{5}, according to [5], we take \( a_p \) as a checkpoint only if \( D(a_p) < Rcd(a_p) \). That is to say, by CSS\textsubscript{5}, we will omit the situation where \( M(a_p) + MPTR\textsubscript{WC}(a_p, a_p) < Rcd(a_p) \leq D(a_p) \). Therefore, CSS\textsubscript{MPTR} is more effective than CSS\textsubscript{5}.

According to [7], at \( a_p \), CSS\textsubscript{6} selects it as a checkpoint if \( M(a_p) < Rcd(a_p) \). However, according to CSS\textsubscript{MPTR}, only if \( M(a_p) + MPTR\textsubscript{WC}(a_p, a_p) < Rcd(a_p) \), we should take \( a_p \) as a checkpoint. That is to say, if \( M(a_p) < Rcd(a_p) \leq M(a_p) + MPTR\textsubscript{WC}(a_p, a_p) \), we need not take \( a_p \) as a checkpoint while CSS\textsubscript{6} does. Therefore, CSS\textsubscript{5} may select some unnecessary checkpoints. Hence, CSS\textsubscript{MPTR} is more efficient than CSS\textsubscript{6}.

6.2. Quantitative evaluation

We now conduct further quantitative analysis so that we can get a specific picture of how CSS\textsubscript{MPTR} is more effective and efficient than CSS\textsubscript{1}, CSS\textsubscript{2}, CSS\textsubscript{3}, CSS\textsubscript{4}, CSS\textsubscript{5} and CSS\textsubscript{6}. However, CSS\textsubscript{1}, CSS\textsubscript{2}, CSS\textsubscript{3} and CSS\textsubscript{4} have some
similarity as they all set static checkpoints at build-time. And CSS₄ is similar to CSS₅ as they both use improper reference values to compare with the activity completion duration. Therefore, we need not compare all of them with CSS₆. We choose CSS₄ and CSS₅ to compare with CSS₆ as they are more representative. We could also compare other strategies with CSS₆. But the corresponding conclusions would be similar.

According to Sections 5 and 6.1, to compare CSS₄ and CSS₅ with CSS₆, we should analyse their unnecessary and omitted verification. According to the consistency definitions in Section 3, primary verification computation is focused on the sum of maximum durations between two activities. Hence, we take each computation of maximum duration addition operation as a verification computation unit. We denote the difference in unnecessary verification computation unit number between CSS₄ and CSS₆ as diff₀(4, MPTR), and that between CSS₅ and CSS₆ as diff₀(6, MPTR). We denote the difference in omitted verification computation unit number between CSS₄ and CSS₆ as diff₁(4, MPTR), and that between CSS₅ and CSS₆ as diff₁(6, MPTR).

We consider a climate modelling grid workflow that may consist of hundreds and thousands of activities and must be time constrained so that the weather forecasting can be broadcast on time [1]. For simplicity, we focus on one fixed-time constraint in it, say FTC(A). We suppose that FTC(A) covers N activities. Since in the real-world grid workflow systems, normally there are many grid workflow instances, we conduct the quantitative analysis in a statistical way. Therefore, for an activity, say aᵢ, we introduce possibility Q for Rcd(aᵢ) ≤ M(aᵢ) + MPTRWC(aᵢ) (0≤Q≤1) and possibility P for Rcd(aᵢ) ≤ M(aᵢ) (0≤P≤1).

For simplicity, we assume that each activity has the same production time. From Table 1, we can see that with Q decreasing, diff₀(4, MPTR) is increasing. This means that the more activities at which we should conduct the fixed-time constraint verification, the more omitted verification based on CSS₄ than based on CSS₆. Therefore, CSS₆ is more effective for fixed-time constraint verification than CSS₄. From Table 2, we can see that diff₁(6, MPTR) is always 0. In fact, according to Section 6.1, CSS₆ does not ignore any checkpoints. Meanwhile, [7] has proven that CSS₆ does not ignore any checkpoints either. Therefore, diff₁(6, MPTR) = 0.

In addition, from Table 1, we can also see that with Q increasing, diff₀(4, MPTR) is increasing. This means that the more activities that can be completed within the sum of the activity mean duration and the minimum WC proportional time redundancy, the more unnecessary verification based on CSS₄ than based on CSS₆. Hence, CSS₆ is more efficient for fixed-time constraint verification than CSS₄. From Table 2, we can see that with Q-P increasing, diff₁(6, MPTR) is increasing. This means that the more activities whose completion durations are greater than their respective mean durations and less than or equal to the activity mean duration plus minimum WC time redundancy, the more unnecessary verification based on CSS₆ than based on CSS₆. Hence, CSS₆ is more efficient for fixed-time constraint verification than CSS₆.

### 7. Conclusions and future work

In this paper, based on the analysis of limitations of existing typical checkpoint selection strategies, a new checkpoint selection strategy has been developed.
Specifically, firstly the concept of minimum proportional time redundancy is introduced. Then, its relationships with fixed-time constraint consistency have been investigated. After that, based on the relationship, the new checkpoint selection strategy has been developed and named CSSMPT (Minimum Proportional Time Redundancy based Checkpoint Selection Strategy). The final comparison and quantitative evaluation have shown that CSSMPT is more effective and efficient for fixed-time constraint verification than existing typical strategies by avoiding the omission of necessary checkpoints and the selection of excess unnecessary checkpoints.

With these contributions, we can further investigate some issues such as temporal exception handling when a fixed-time constraint is violated at a checkpoint.

Acknowledgements

The work reported in this paper is partly supported by Swinburne Vice Chancellor’s Strategic Research Initiative Grant and by ARC Linkage Project under grant No. LP0562500.

References


