Adaptive Selection of Necessary and Sufficient Checkpoints for Dynamic Verification of Temporal Constraints in Grid Workflow Systems

JINJUN CHEN, YUN YANG
Swinburne University of Technology

In grid workflow systems, a checkpoint selection strategy is responsible for selecting checkpoints for conducting temporal verification at run-time execution stage. Existing representative checkpoint selection strategies often select some unnecessary checkpoints and omit some necessary ones because they cannot adapt to the dynamics and uncertainty of run-time activity completion duration. In this paper, based on the dynamics and uncertainty of run-time activity completion duration, we develop a novel checkpoint selection strategy that can adaptively select not only necessary but also sufficient checkpoints. Specifically, we introduce a new concept of minimum time redundancy as a key reference parameter for checkpoint selection. An important feature of minimum time redundancy is that it can adapt to the dynamics and uncertainty of run-time activity completion duration. We develop a method on how to achieve minimum time redundancy dynamically along grid workflow execution, and investigate its relationships with temporal consistency. Based on the method and the relationships, we present our strategy and rigorously prove its necessity and sufficiency. The simulation evaluation further experimentally demonstrates such necessity and sufficiency and its significant improvement on checkpoint selection over other representative strategies.

---

1. INTRODUCTION

In a grid architecture, a grid workflow management system is a type of high-level grid middleware which is supposed to support modelling, redesign and execution of large-scale sophisticated scientific and business processes in a variety of e-science and e-business applications such as climate modelling, astrophysics, high energy physics, international finance and insurance [Abramson et al. 2004; Cao et al. 2003; Foster et al. 2002, Marinescu 2002]. Then, they are instantiated at run-time instantiation stage by an instantiation grid service [Amin et al. 2004; Cybok 2004]. Finally, they are executed at run-time execution stage by facilitating the computing and resource sharing power of underlying grid infrastructure. The execution is coordinated between grid services by the grid workflow engine that itself is a high-level grid service [Cybok 2004; Huang 2003].

1.1 Temporal Constraints

In reality, complex scientific or business processes are normally time constrained [Al-Ali et al. 2004; Buyya et al. 2005; Yu et al. 2005]. Accordingly, temporal constraints are often enforced when they are modelled as grid workflow specifications. The types of temporal constraints mainly include upper bound, lower bound and fixed-time [Chen and Yang 2005b; 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 [Eder et al. 1999]. 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 [Eder et al. 1999]. A fixed-time constraint at an activity is an absolute time value by which the activity must be completed [Chen and Yang 2005b; Li et al. 2004]. For example, a climate modelling grid workflow must be completed by the scheduled time [Abramson et al. 2004], say 7:00pm, so that the weather forecast can be broadcast at a later time. Here, 7pm is a fixed-time constraint. Some references have also addressed two other types of temporal constraints: deadline and fixed-date [Eder et al. 1999; Marjanovic and Orlowska 1999]. [Marjanovic and Orlowska 1999] divides deadline constraints into relative and absolute deadline constraints. A relative deadline constraint is a time value which is relative to the start time of the grid workflow. An absolute deadline constraint is an absolute time value such as 7:00pm, May 1. Apparently, a relative deadline constraint is actually an upper bound constraint whose start activity is exactly the start activity of the whole grid workflow, while an absolute deadline constraint is just a fixed-time constraint. Hence, in this paper, we do not discuss deadline constraints separately. In addition, according to [Eder et al. 1999], a fixed-date constraint at an activity is an absolute time value by which the activity must be completed. Apparently, a fixed-date constraint is just a fixed-time constraint. Hence, we do not discuss fixed-date constraints separately either.

1.2 Temporal Verification and Checkpoint Selection

After temporal constraints are set, temporal verification must be conducted so that we can identify and handle temporal violations in time in order to ensure the overall temporal correctness. At build-time and run-time instantiation stages, temporal verification is static because there are no any specific execution times. Each temporal constraint needs to be verified only once with the consideration of all covered activities. Therefore, we need not 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 temporal 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 allowed time intervals. So where should we conduct the temporal verification? The activities at which we conduct the verification are called ***checkpoints*** [Eder et al. 1999; Marjanovic and Orlowska 1999; Zhuge et al. 2001]. This is the topic of the research field on ***CSS*** (Checkpoint Selection Strategies) [Eder et al. 1999; Marjanovic and Orlowska 1999; Zhuge et al. 2001].

Some representative checkpoint selection strategies have been proposed [Eder et al. 1999; Marjanovic and Orlowska 1999; Zhuge et al. 2001; Chen et al. 2004; Chen and Yang 2005a; Chen and Yang 2005c]. They are detailed in Section 3. However, they cannot adapt to the dynamics and uncertainty of run-time activity completion duration. Consequently, they often suffer from the limitations of selecting unnecessary checkpoints and omitting necessary ones. Unnecessary checkpoints will result in unnecessary temporal verification while omitted checkpoints will cause necessary verification omitted. Clearly, neither is desirable. Therefore, in this paper, based on the dynamics and uncertainty of run-time activity completion duration, we develop a new checkpoint

selection strategy that can adaptively select not only necessary but also sufficient checkpoints.

1.3 Paper Organisation

The remainder of the paper is organised as follows. In Section 2, we represent some time attributes of grid workflows. In Section 3, we detail the related work and problem analysis for checkpoint selection. Then, in Section 4, we introduce a new concept of minimum time redundancy which will serve as a key reference parameter for our strategy. An important feature of minimum time redundancy is that it can adapt to the dynamics and uncertainty of run-time activity completion duration. We also develop a method on how to obtain minimum time redundancy dynamically along grid workflow execution. After that, in Section 5, we investigate the relationships between minimum time redundancy and temporal consistency in depth. Based on these relationships, we present our new strategy and rigorously prove its necessity and sufficiency for checkpoint selection. In Section 6, we perform a simulation to demonstrate the necessity and sufficiency of our strategy and the significant improvement on checkpoint selection over other representative strategies. Finally, in Section 7, we conclude our contributions and point out future work.

2. TIMED GRID WORKFLOW REPRESENTATION

According to [Li et al. 2003; Marjanovic and Orlowska 1999], 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 [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 grid workflow graph is structurally acyclic[1]. Here we assume that a grid 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 1998; Aalst 2000; Sadiq and Orlowska 2000].

2.1 Activity Time Attributes and Temporal Constraints

To represent activity time attributes in a grid workflow, we borrow some concepts from [Chinn and Madey 2000; Eder et al. 1999; Marjanovic and Orlowska 1999] such as the maximum, mean or minimum duration as a basis. We denote the ith activity of a grid 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 ($i<j$), the maximum duration, mean duration and minimum duration between them are denoted 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 [Liu 1998; 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 $d(a_i) \leq M(a_i) \leq D(a_i)$ and $d(a_i) \leq R(a_i) \leq D(a_i)$.

---

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

3

Regarding the representation of temporal constraints, according to Section 1.1, conceptually a lower bound constraint is symmetrical to an upper bound constraint. For example, concerning a lower bound constraint, we often check whether the duration between its start and end activity is "$\geq$" its value. For an upper bound constraint, we often check whether the duration between its start and end activity is "$\leq$" its value. As to a fixed-time constraint, we can view the first activity of a grid workflow as its start activity. Then, the 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. As such, in this paper, we focus on upper bound constraints only. The corresponding checkpoint selection discussion and results can be equally applied to lower bound and fixed-time constraints. Correspondingly, if there is a path from a_i to a_j ($i<j$) 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 convenience of the discussion, we only consider 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.

2.2 Temporal Consistency States

Besides the time attributes represented in Section 2.1, [Chen and Yang 2005b] has identified and defined 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 minimum time redundancy concept to be addressed in Section 4 is related to run-time instantiation and execution stages, we summarise the definitions for these two stages here. The definitions for build-time stage and the detailed discussion can be found in [Chen and Yang 2005b; Eder et al. 1999].

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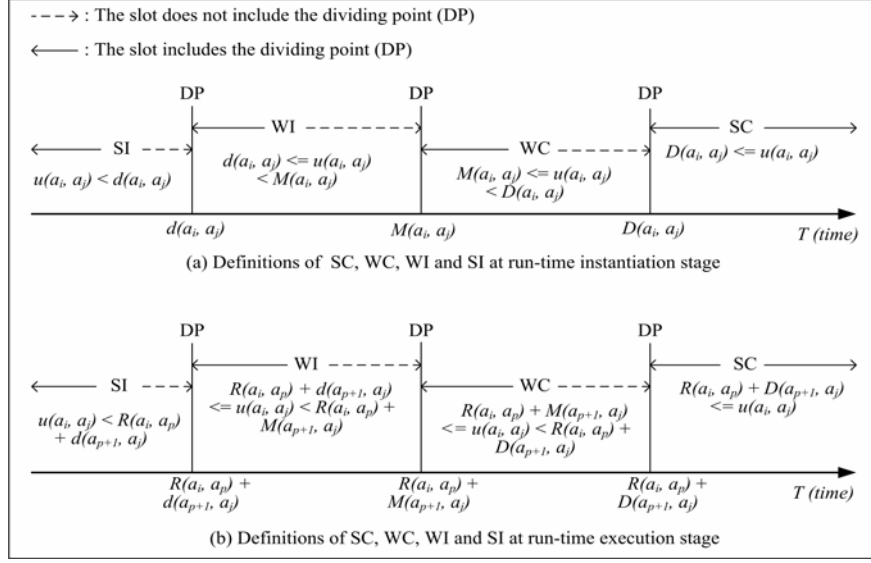
Fig. 1. Definitions of SC, WC, WI and SI at run-time instantiation and execution stages

According to [Chen and Yang 2005b], along grid workflow execution, for SC, we need not do anything as the corresponding upper bound constraints can be kept. For WC, by utilising the possible time redundancy of succeeding activity execution, 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 is triggered to adjust them to SC or WC. Specific exception handling methods can be found in [Hagen and Alonso 2000].

Since WI and SI are adjusted to SC or WC by the 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.

3. RELATED WORK AND PROBLEM ANALYSIS

3.1 Existing Representative Checkpoint Selection Strategies

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.

5

- CSS_6: [Chen and Yang 2005a] 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 2005c] 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.

3.2 Problem Analysis

In this section, we analyse the problem of $CSS_1 \sim CSS_7$ on checkpoint selection. Considering a climate modelling grid workflow which contains 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 [Abramson et al. 2004], we take one of its segments as the example to reason about the problem analysis. Suppose the segment is the kth one and involves two upper bound constraints denoted as U_1 and U_2. We depict them and attach some time values in Figure 2. The time unit is hour. Figure 2 contains a selective structure which has two branches, i.e. Branch 1 and Branch 2. There can be many execution instances for Figure 2. We use some instances which affect SC of U_1 and U_2. The corresponding discussion for WC is similar.
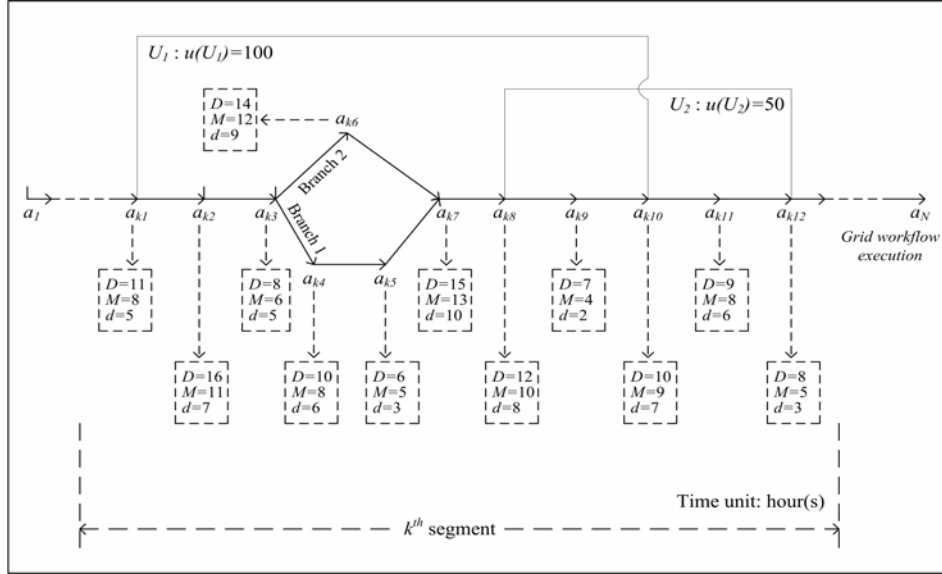

Fig. 2. A sample segment of a climate modelling grid workflow

CSS_1 and CSS_2 select every activity as a checkpoint. We consider an execution instance where the execution goes Branch 1 and $R(a_{k1})=9$. Then, at a_{k1}, we have (1) below.

$$R(a_{k1})+D(a_{k2})+D(a_{k3})+D(a_{k4})+D(a_{k5})+D(a_{k7})+D(a_{k8})+D(a_{k9})+D(a_{k10})=9+16+8+10+6$$
$$+15+12+7+10 = 93 < u(U_1) \quad (1)$$

According to Definition 2, (1) means that U_1 is of SC. That is to say, we actually need not select a_{k1} as a checkpoint for conducting temporal verification because current U_1 can be kept and hence there are no any temporal violations. Therefore, CSS_1 and CSS_2 may select some unnecessary checkpoints. Since they take every activity as a checkpoint, they do not omit any necessary ones.

6

We now discuss CSS_3 and CSS_4. In Figure 2, CSS_3 only selects a_1 (the start activity) and a_{k3} (a decision activity) as checkpoints. We consider an execution case where the execution goes Branch 1, $R(a_{k1})$ =11 and $R(a_{k2})$=22. Run-time activity execution duration is dynamic. Hence, it is possible for $R(a_{k2})$ to exceed $D(a_{k2})$ which is 16. In that case, at a_{k2}, we have (2) below.

$$R(a_{k1})+R(a_{k2})+D(a_{k3})+D(a_{k4})+D(a_{k5})+D(a_{k7})+D(a_{k8})+D(a_{k9})+D(a_{k10})=11+22+8+10+$$
$$6+15+12+7+10=101 > u(U_1) \quad (2)$$

According to Definition 2, (2) means that U_1 is violated and is not of SC. That is to say, we should select a_{k2} as a checkpoint so that we can identify the temporal violation in time. However, CSS_3 does not select it as a checkpoint. Hence, CSS_3 may omit some necessary checkpoints. We further consider another execution case where the execution goes Branch 1, $R(a_{k1})$=11, $R(a_{k2})$=14 and $R(a_{k3})$=6. Then, at a_{k3}, we have (3) below.

$$R(a_{k1})+R(a_{k2})+R(a_{k3})+D(a_{k4})+D(a_{k5})+D(a_{k7})+D(a_{k8})+D(a_{k9})+D(a_{k10})=11+14+6+10$$
$$+6+15+12+7+10=91 < u(U_1) \quad (3)$$

According to Definition 2, (3) means that U_1 is of SC. That is to say, we actually need not select a_{k3} as a checkpoint. However, CSS_3 does select it as a checkpoint. Hence, CSS_3 may select some unnecessary checkpoints. Regarding CSS_4, it defines some static activity points as checkpoints at build-time stage. Suppose that it defines a_{k3}, a_{k5} and a_{k8} as checkpoints. Then, the same problem of CSS_3 on checkpoint selection will happen. The corresponding reasoning is similar to the above for CSS_3.

We now move on to CSS_5, CSS_6 and CSS_7. We consider an execution case of Figure 2 where the execution goes Branch 2, $R(a_{k1})$=9, $R(a_{k2})$=14, $R(a_{k3})$=6, $R(a_{k6})$=11, $R(a_{k7})$=13 and $R(a_{k8})$=15. Then at a_{k8}, we have (4) below.

$$R(a_{k1})+R(a_{k2})+R(a_{k3})+R(a_{k6})+R(a_{k7})+R(a_{k8})+D(a_{k9})+D(a_{k10})=9+14+6+11+13+15+7+$$
$$10=85 < u(U_1) \quad (4)$$

According to Definition 2, (4) means that U_1 is of SC. That is to say, we need not select a_{k8} as a checkpoint. However, since $R(a_{k8})>D(a_{k8})=12$ and $R(a_{k8})>M(a_{k8})=10$, both CSS_5 and CSS_6 select a_{k8} as a checkpoint. Hence, CSS_5 and CSS_6 may select some unnecessary checkpoints. [Chen et al. 2004; Chen and Yang 2005a] have proved that we need not select activity a_i as a checkpoint if $R(a_i) \leq D(a_i)$ or $R(a_i) \leq M(a_i)$. Accordingly, CSS_5 and CSS_6 do not omit any necessary checkpoints. Regarding CSS_7, in [Chen and Yang 2005c], CSS_7 introduces minimum proportional time redundancy (MPTR) to each activity. We denote that at a_{k8} as $MPTR(a_{k8})$. According to [Chen and Yang 2005c], U_1 has a build-time static time redundancy that is shown in (5) below.

$$u(U_1)-[D(a_{k1})+D(a_{k2})+D(a_{k3})+D(a_{k6})+ D(a_{k7})+D(a_{k8})+D(a_{k9})+D(a_{k10})] = 7 \quad (5)$$

U_2 also has a build-time static time redundancy that is shown in (6) below.

$$u(U_2)-[D(a_{k8})+D(a_{k9})+D(a_{k10})+D(a_{k11})+D(a_{k12})] = 4 \quad (6)$$

CSS_7 allocates the build-time static time redundancy of U_1 to a_{k8} in the proportion of $D(a_{k8})-M(a_{k8})$ among all $D(a_{ki})-M(a_{ki})$ (i=1~3, 6~10). Then, a_{k8} holds a static time quota. According to CSS_7, we first sort all $D(a_{ki})-M(a_{ki})$ (i=1~3, 6~10) in ascending order and we then get a sorting list. We denote items in the sorting list as L_1, L_2, ... , L_j. Then if $D(a_{k8})-M(a_{k8})$ is ranked number m, i.e. L_m ($1 \leq m \leq j$), the time quota allocated to a_{k8} is equal

to the build-time static time redundancy of U_1 multiplied by $\{L_{j-m+1} / \sum_{i=1\sim3, 6\sim10}[D(a_{ki}) - M(a_{ki})]\}$.

Based on this, we can figure out that j=8, m=5, L_m=$D(a_{k8})$-$M(a_{k8})$=2, L_{j-m+1}=$D(a_{k7})$-$M(a_{k7})$=2, and the sum of all $D(a_{ki})$-$M(a_{ki})$ (i=1~3, 6~10) is 20. Hence, the static time quota allocated to a_{k8} is 7*(2/20), i.e. 0.7. Similarly, we can figure out that the static time quota allocated to a_{k8} from the static time redundancy of U_2 is 0.5. According to CSS_7 [Chen and Yang 2005c], $MPTR(a_{k8})$ is equal to the minimum one of the two quotas, i.e. 0.5. CSS_7 selects a_{k8} as a checkpoint if $R(a_{k8})>M(a_{k8})+MPTR(a_{k8})$. In this case, $R(a_{k8})$=15

and $M(a_{k8})+MPTR(a_{k8})=10+0.5=10.5$. Hence, we do have $R(a_{k8})>M(a_{k8})+MPTR(a_{k8})$. That is to say, CSS_7 selects a_{k8} as a checkpoint although it is not necessary. Therefore, CSS_7 may also select some unnecessary checkpoints. [Chen and Yang 2005c] has proved that we need not select activity a_i as a checkpoint if $R(a_i) \le M(a_i)+MPTR(a_i)$. Accordingly, CSS_7 does not omit any necessary checkpoints.

In summary, CSS_1 and CSS_2 may select some unnecessary checkpoints although they do not omit any necessary ones. CSS_3 and CSS_4 may select some unnecessary checkpoints and omit some necessary ones. CSS_5, CSS_6 and CSS_7 may select some unnecessary checkpoints without omitting any necessary ones.

3.3 Further Anatomy of the Problem

We conclude that the radical reason for the problem of existing representative checkpoint selection strategies is that they cannot fully adapt to the dynamics and uncertainty of run-time activity completion duration. The further explanation is presented below.

Grid workflow execution environments are very dynamic since they normally encompass multiple administrative domains (organisations) over a wide area network [Deelman et al. 2003; Han 1998; Klingemann 1999a; Klingemann 1999b]. A grid service may need to simultaneously serve the execution of a number of grid workflow activities from many grid workflow instances while sometimes many grid services are available for one activity execution [Buyya et al. 2005; Foster 2005]. As such, the completion duration of a grid workflow activity is highly dynamic and uncertain [Chen and Yang 2005b; Reichert et al. 1999; Rinderle et al. 2004]. We may need to verify temporal constraints at some activities while at others we do not need to. That is to say, the corresponding checkpoint selection should have the capability of adapting to such dynamics and uncertainty.

However, CSS_1, CSS_2, CSS_3 and CSS_4 do not have this capability because they predefine checkpoints before grid workflow execution. Some of the activities which are predefined as checkpoints may be able to be completed within allowed time intervals and consequently their executions will not impact the consistency of any temporal constraints. That is to say, at such activities, we actually need not conduct any temporal verification, i.e. we should not have taken them as checkpoints. Therefore, CSS_1, CSS_2, CSS_3 and CSS_4 may select some unnecessary checkpoints. In addition, some of other activities which are not predefined as checkpoints may be completed exceeding the allowed time intervals and consequently their executions may impact the consistency of some temporal constraints. Hence, at such activities, we need to conduct temporal verification, i.e. we should have taken them as checkpoints. Therefore, although CSS_1 and CSS_2 do not omit any necessary checkpoints as they select every activity as a checkpoint, CSS_3 and CSS_4 may omit some.

CSS_5, CSS_6 and CSS_7 have improved CSS_1, CSS_2, CSS_3 and CSS_4 in their capability of adapting to the dynamics and uncertainty of run-time activity completion duration. They utilise the activity completion duration during their checkpoint selection process. However, they have not done it fully because their key reference parameters are static. These parameters are maximum duration, mean duration and minimum proportional time redundancy. According to [Chen et al. 2004; Chen and Yang 2005a, Chen and Yang 2005c] and Section 3.2, all of them are statically set at build-time stage without taking into consideration of run-time activity completion duration. Therefore, CSS_5, CSS_6 and CSS_7 still cannot fully adapt to the dynamics and uncertainty of run-time activity completion duration. In Section 3.2, we have seen that they still suffer from the limitation of selecting unnecessary checkpoints.

Regarding the above limitations of the existing representative checkpoint selection strategies, we may raise the research question: "Can we develop a checkpoint selection

strategy that can adaptively select necessary yet sufficient checkpoints on the fly along grid workflow execution?". This question and some concepts have been mentioned briefly in [Chen and Yang 2006], but no sufficient details were provided there. In this paper, we answer the question comprehensively by presenting such a strategy in detail. Our fundamental idea is that based on CSS_5, CSS_6 and CSS_7, we further take activity completion duration into the construction of key reference parameters. For this purpose, we introduce a new concept of minimum time redundancy which serves as a key reference parameter for our checkpoint selection. An important feature of minimum time redundancy is that it is based on activity completion duration and consequently can adapt to the dynamics and uncertainty of run-time activity completion duration. The detailed discussion of minimum time redundancy is presented in the next section.

4. MINIMUM TIME REDUNDANCY

In this section, we discuss minimum time redundancy in detail. According to Section 2, since checkpoint selection is actually for SC and WC upper bound constraint verification, minimum time redundancy consists of minimum SC and WC time redundancy. The former is for SC upper bound constraints and the later is for WC ones.

First, we introduce the concept of SC and WC time redundancy from one upper bound constraint in Section 4.1. Then, we introduce minimum SC and WC time redundancy from multiple upper bound constraints in Section 4.2. After that, in Section 4.3, we discuss how to obtain minimum SC and WC time redundancy dynamically along grid workflow execution.

4.1 SC and WC Time Redundancy

At run-time execution stage, we consider a SC upper bound constraint, say $U(a_i, a_j)$, as shown in Figure 3. At activity point a_p between a_i and a_j, according to Definition 2, we have $R(a_i, a_p) + D(a_{p+1}, a_j) \leq u(a_i, a_j)$. Clearly, there is a time difference: $u(a_i, a_j) - [R(a_i, a_p) + D(a_{p+1}, a_j)]$. The difference means that if the succeeding activity execution can be controlled within the difference, $U(a_i, a_j)$ can still be kept as SC regardless whether the execution consumes more time than scheduled. Correspondingly, we define this time difference as SC time redundancy of $U(a_i, a_j)$ at activity point a_p, and denote it as $TR_{SC}(U(a_i, a_j), a_p)$ (TR_{SC}: SC Time Redundancy). Hence, we have Definition 3 below.

Definition 3. At activity point a_p between a_i and a_j ($i<j$), let $U(a_i, a_j)$ be of SC (as shown in Figure 3). 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 $TR_{SC}(U(a_i, a_j), a_p)$:

$$TR_{SC}(U(a_i, a_j), a_p) = u(a_i, a_j) - [R(a_i, a_p) + D(a_{p+1}, a_j)].$$
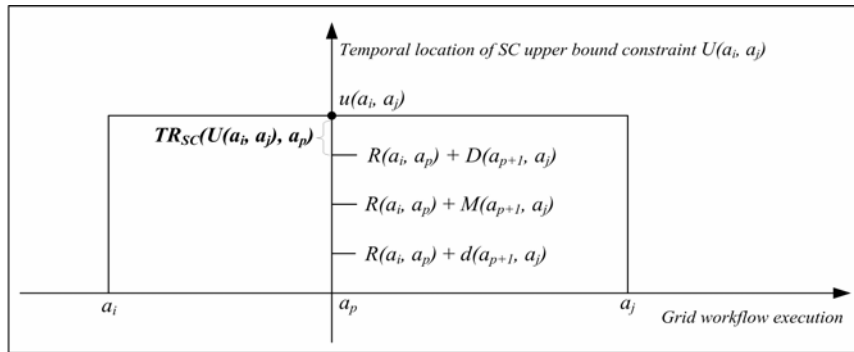

Fig. 3. SC $U(a_i, a_j)$ and its SC time redundancy at a_p

For a WC upper bound constraint, say $U(a_k, a_l)$, similarly we have WC time redundancy and define it in Definition 4 below.

Definition 4. At activity point a_p between a_k and $a_l (k<l)$, let $U(a_k, a_l)$ be of WC (as shown in Figure 4). 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 $TR_{WC}(U(a_k, a_l), a_p)$:

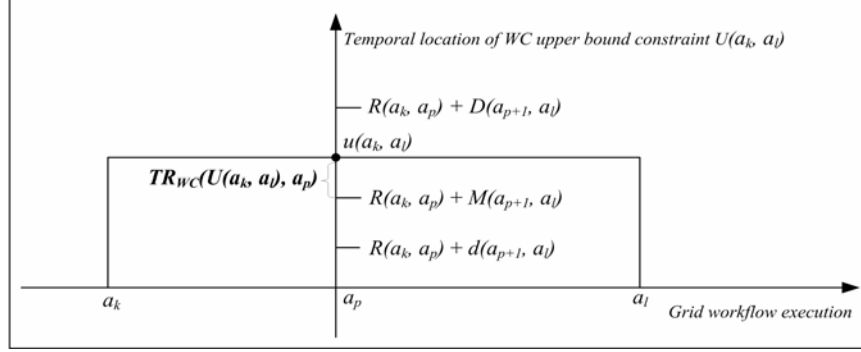$$TR_{WC}(U(a_k, a_l), a_p) = u(a_k, a_l) - [R(a_k, a_p) + M(a_{p+1}, a_l)].$$

Fig. 4. WC $U(a_k, a_l)$ and its WC time redundancy at a_p

For example, we consider an execution instance of Figure 2 where the execution goes Branch 1, $R(a_{k1})$=9, $R(a_{k2})$=11, $R(a_{k3})$=7, $R(a_{k4})$=9, $R(a_{k5})$=5, $R(a_{k7})$=13 and $R(a_{k8})$=10. Then at a_{k8}, according to Definition 2, U_1 and U_2 are of SC. Correspondingly, we have SC time redundancies as shown in (7) and (8) below.

$$TR_{SC}(U_1, a_{k8})=u(U_1)-[R(a_{k1})+R(a_{k2})+R(a_{k3})+R(a_{k4})+R(a_{k5})+R(a_{k7})+R(a_{k8})+D(a_{k9}, a_{k10})]=100-[9+11+7+9+5+13+10+7+10]=19 \quad (7)$$

$$TR_{SC}(U_2, a_{k8})=u(U_2)-[R(a_{k8}, a_{k8})+D(a_{k9}, a_{k12})]=50-[10+7+10+9+8]=6 \quad (8)$$

If we consider another execution instance of Figure 2 where the execution goes Branch 1, but $R(a_{k1})$=11, $R(a_{k2})$=16, $R(a_{k3})$=8, $R(a_{k4})$=10, $R(a_{k5})$=6, $R(a_{k7})$=14 and $R(a_{k8})$=19, then at a_{k8}, according to Definition 2, U_1 and U_2 are of WC. Correspondingly, we have WC time redundancies as shown in (9) and (10) below.

$$TR_{WC}(U_1, a_{k8})=u(U_1)-[R(a_{k1})+R(a_{k2})+R(a_{k3})+R(a_{k4})+R(a_{k5})+R(a_{k7})+R(a_{k8})+M(a_{k9}, a_{k10})]= 100-[11+16+8+10+6+14+19+4+9]=3 \quad (9)$$

$$TR_{WC}(U_2, a_{k8})= u(U_2)-[R(a_{k8}, a_{k8})+M(a_{k9}, a_{k12})]=50-[19+4+9+8+5]=5 \quad (10)$$

4.2 Minimum SC and WC Time Redundancy

We now consider multiple SC or WC upper bound constraints that cover a_p, i.e. a_p is between the start activities and end activities of those upper bound constraints. Based on Definitions 3 and 4, we introduce minimum SC and WC time redundancy.

Definition 5 (*Minimum SC Time Redundancy*). Let U_1, U_2, ... , 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, ... , U_N, and is denoted as $MTR_{SC}(a_p)$ (MTR_{SC}: SC Minimum Time Redundancy):

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

Definition 6 (*Minimum WC Time Redundancy*). Let U_1, U_2, ... , 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, ... , U_N, and is denoted as $MTR_{WC}(a_p)$ (MTR_{WC}: WC Minimum Time Redundancy):

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

For example, we consider the two execution instances used in Section 4.1 again. For the execution instance where U_1 and U_2 are of SC, at a_{k8}, we have (11) below.

$$MTR_{SC}(a_{k8})=Min\{TR_{SC}(U_1, a_{k8}), TR_{SC}(U_2, a_{k8})\}=Min\{19, 6\}=6 \qquad (11)$$

For the execution instance where U_1 and U_2 are of WC, we have (12) below.

$$MTR_{WC}(a_{k8})=Min\{TR_{WC}(U_1, a_{k8}), TR_{WC}(U_2, a_{k8})\}=Min\{3, 5\}=3 \qquad (12)$$

According to Definitions 5 and 6, 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.

In addition, we normally have $M(a_p) + MTR_{WC}(a_{p-1}) < D(a_p) + MTR_{SC}(a_{p-1})$. The reason is simple: if $M(a_p) + MTR_{WC}(a_{p-1}) \geqslant D(a_p) + MTR_{SC}(a_{p-1})$, then, since the upper bound constraint of $MTR_{SC}(a_{p-1})$ is of SC, the upper bound constraint of $MTR_{WC}(a_{p-1})$ must also be of SC; but the upper bound constraint of $MTR_{WC}(a_{p-1})$ is actually of WC.

4.3 Dynamic Obtaining of Minimum SC and WC Time Redundancy

Along grid workflow execution, at a_p, an intuitive method for obtaining $MTR_{SC}(a_p)$ and $MTR_{WC}(a_p)$ is to compute and compare all SC and WC time redundancies. However, this method is not efficient as it will cause too much extra computation. Hence, we develop a more efficient method and denote it as DOMTR (Dynamic Obtaining of Minimum Time Redundancy). We describe the detailed working process of DOMTR in Appendix A.

In brief, from Appendix A, we can see that DOMTR works at run-time instantiation and execution stages. At run-time instantiation stage, DOMTR sets up some initial values. Then, at run-time execution stage, DOMTR keeps minimum SC and WC time redundancy along grid workflow execution. At each activity, DOMTR adjusts minimum SC and WC time redundancy in time according to the dynamic change of activity completion duration. As such, minimum SC and WC time redundancy can adapt to dynamics and uncertainty of run-time activity completion duration.

Compared to the intuitive method, DOMTR involves far less extra computation. According to Appendix A, at run-time instantiation stage, DOMTR sets up some initial values by directly using the corresponding computation results from temporal verification. Since temporal verification must be conducted at run-time instantiation stage regardless whether or not we select some checkpoints for run-time execution stage verification, DOMTR does not incur any extra computation. At run-time execution stage, DOMTR computes minimum SC and WC time redundancy on the fly along grid workflow execution. Basically, the extra computation we need is one or two subtractions or comparisons at each activity covered by one or more 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 2005b; Marjanovic and Orlowska 1999; Zhuge et al. 2001], such one or two subtractions or comparisons would be negligible.

5. CHECKPOINT SELECTION BASED ON MINIMUM SC AND WC TIME REDUNDANCY

5.1 Relationships between Minimum SC & WC Time Redundancy and SC, WC, WI & SI

At the run-time execution stage, at activity point a_p, we discuss relationships between $MTR_{SC}(a_{p-1})$ & $MTR_{WC}(a_{p-1})$ and SC, WC, WI & SI. We first depict these relationships in Figure 5.
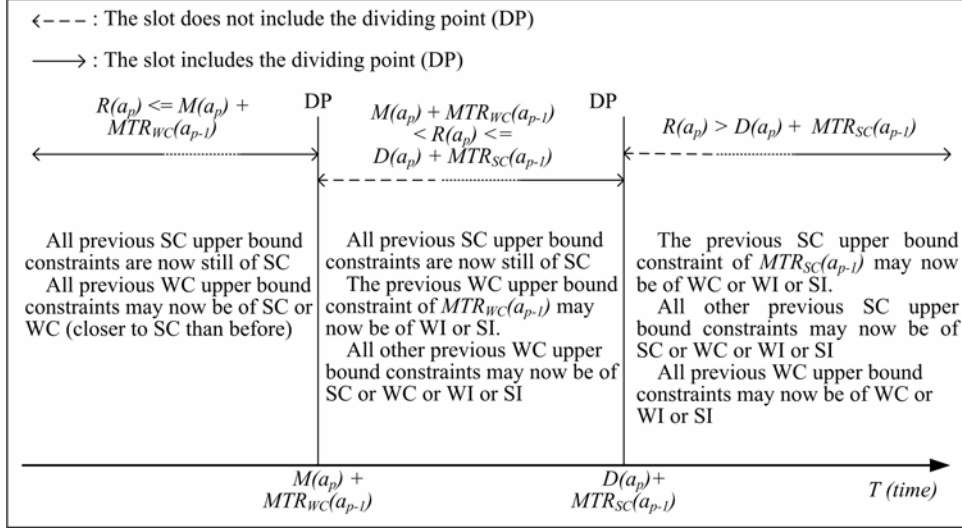
Fig. 5. Relationships between minimum SC and WC time redundancy and SC, WC, WI & SI

As shown in Figure 5, there are three relationships. We now further explain them. For this purpose, we draw three theorems: Theorems 1, 2 and 3. Theorem 1 is used to support the relationships where $R(a_p) > D(a_p) + MTR_{SC}(a_{p-1})$. Theorem 2 is used to support the relationships where $M(a_p) + MTR_{WC}(a_{p-1}) < R(a_p) \leq D(a_p) + MTR_{SC}(a_{p-1})$. And Theorem 3 is used to support the relationships where $R(a_p) \leq M(a_p) + MTR_{WC}(a_{p-1})$.

Theorem 1. At activity point a_p, if $R(a_p) > D(a_p) + MTR_{SC}(a_{p-1})$, then:
1) all previous WC upper bound constraints now cannot be of SC and may be of WC, WI or SI;
2) the previous SC upper bound constraint whose minimum SC time redundancy at a_{p-1} is $MTR_{SC}(a_{p-1})$ now cannot be of SC and may be of WC, WI or SI; and
3) all other previous SC upper bound constraints may now be of SC, WC, WI or SI.

Proof: 1) Suppose $U(a_k, a_l)$ is a previous WC upper bound constraint, i.e. it is of WC before execution of a_p ($k \leq p \leq l$). Then, according to Definition 2, we have (13) below.

$$u(a_k, a_l) < R(a_k, a_{p-1}) + D(a_p, a_l) \tag{13}$$

Besides, since $R(a_p) > D(a_p) + MTR_{SC}(a_{p-1})$, $D(a_p) < R(a_p)$. Together with (13), then $u(a_k, a_l) < R(a_k, a_{p-1}) + D(a_p, a_l) = R(a_k, a_{p-1}) + D(a_p) + D(a_{p+1}, a_l) < R(a_k, a_{p-1}) + R(a_p) + D(a_{p+1}, a_l) = R(a_k, a_p) + D(a_{p+1}, a_l)$. Hence, we have (14) below.

$$u(a_k, a_l) < R(a_k, a_p) + D(a_{p+1}, a_l) \tag{14}$$

According to Definition 2, (14) means that $U(a_k, a_l)$ can not be of SC after execution of a_p.

In addition, since $R(a_p) > D(a_p) + MTR_{SC}(a_{p-1})$, $R(a_k, a_{p-1}) + M(a_p, a_l) = R(a_k, a_{p-1}) + M(a_p) + M(a_{p+1}, a_l) \leq R(a_k, a_{p-1}) + D(a_p) + M(a_{p+1}, a_l) < R(a_k, a_{p-1}) + R(a_p) - MTR_{SC}(a_{p-1}) + M(a_{p+1}, a_l) = R(a_k, a_p) + M(a_{p+1}, a_l) - MTR_{SC}(a_{p-1})$. Hence, we have (15) below.

$$R(a_k, a_{p-1}) + M(a_p, a_l) < R(a_k, a_p) + M(a_{p+1}, a_l) - MTR_{SC}(a_{p-1}) \tag{15}$$

Meanwhile, because $U(a_k, a_l)$ is previously of WC, according to Definition 2, we have (16) below.

$$R(a_k, a_{p-1}) + M(a_p, a_l) \leq u(a_k, a_l) \tag{16}$$

However, (15) and (16) are insufficient to judge whether (17) below holds or not.

$$R(a_k, a_p) + M(a_{p+1}, a_l) \leq u(a_k, a_l) \tag{17}$$

If (17) holds, $U(a_k, a_l)$ is of WC again. However, depending on specific $MTR_{SC}(a_{p-1})$, (17) may or may not hold. Similarly, we may or may not have $R(a_k, a_p) + d(a_{p+1}, a_l) \leq u(a_k, a_l) < R(a_k, a_p) + M(a_{p+1}, a_l)$ and $u(a_k, a_l) < R(a_k, a_p) + d(a_{p+1}, a_l)$. Therefore, according to

12

Definition 2, depending on specific $MTR_{SC}(a_{p-1})$, after execution of a_p, $U(a_k, a_l)$ may be of WC, WI or SI.

2) Suppose the previous SC upper bound constraint corresponding to $MTR_{SC}(a_{p-1})$ is $U(a_i, a_j)$. Then, according to Definitions 3 and 5, we have (18) below.

$$MTR_{SC}(a_{p-1}) = u(a_i, a_j) - [R(a_i, a_{p-1}) + D(a_p, a_j)] \qquad (18)$$

Together with $R(a_p) > D(a_p) + MTR_{SC}(a_{p-1})$, then $u(a_i, a_j) - [R(a_i, a_{p-1}) + D(a_p, a_j)] < R(a_p) - D(a_p)$, i.e. $u(a_i, a_j) < R(a_p) - D(a_p) + [R(a_i, a_{p-1}) + D(a_p, a_j)] = R(a_i, a_p) + D(a_{p+1}, a_j)$. Hence, we have (19) below.

$$u(a_i, a_j) < R(a_i, a_p) + D(a_{p+1}, a_j) \qquad (19)$$

According to Definition 2, (19) means that $U(a_i, a_j)$ can not be of SC after execution of a_p.

In addition, Similar to 1), we can prove that depending on specific $MTR_{SC}(a_{p-1})$, $U(a_i, a_j)$ may be of WC, WI or SI after execution of a_p.

3) Suppose $U(a_i, a_j)$ is a previous SC upper bound constraint, i.e. it is of SC before execution of a_p ($i \leq p \leq j$). We also suppose $U(a_i, a_j)$ is not the one whose minimum SC time redundancy at a_{p-1} is $MTR_{SC}(a_{p-1})$. Then, according to Definitions 3 and 5, we have (20) below.

$$MTR_{SC}(a_{p-1}) \leq TR_{SC}(U(a_i, a_j), a_p) \qquad (20)$$

Together with $R(a_p) > D(a_p) + MTR_{SC}(a_{p-1})$, it is insufficient to decide whether $TR_{SC}(U(a_i, a_j), a_p) + D(a_p) < R(a_p)$. If $TR_{SC}(U(a_i, a_j), a_p) + D(a_p) < R(a_p)$, then, similar to 2), we can prove that $U(a_i, a_j)$ cannot be of SC after execution of a_p. But if $R(a_p) \leq TR_{SC}(U(a_i, a_j), a_p) + D(a_p)$, then, similar to 1), we can prove that depending on specific $TR_{SC}(U(a_i, a_j), a_p)$, after execution of a_p, $U(a_i, a_j)$ may be of SC, WC, WI or SI.

Thus, in overall terms, the theorem holds. ▮

Theorem 2. At activity point a_p, if $M(a_p) + MTR_{WC}(a_{p-1}) < R(a_p) \leq D(a_p) + MTR_{SC}(a_{p-1})$, then:

1) all previous SC upper bound constraints are now still of SC;
2) the previous WC upper bound constraint whose minimum WC time redundancy at a_{p-1} is $MTR_{WC}(a_{p-1})$ now cannot be of WC and SC, and may now be of WI or SI; and
3) all other previous WC upper bound constraints may now be of SC, WC, WI or SI.

Proof: 1) Suppose $U(a_i, a_j)$ is of SC before execution of a_p. According to Definition 2, we have (21) below.

$$R(a_i, a_{p-1}) + D(a_p, a_j) \leq u(a_i, a_j) \qquad (21)$$

Together with $R(a_p) \leq D(a_p) + MTR_{SC}(a_{p-1})$, then $R(a_i, a_p) + D(a_{p+1}, a_j) = R(a_i, a_{p-1}) + R(a_p) + D(a_{p+1}, a_j) \leq R(a_i, a_{p-1}) + MTR_{SC}(a_{p-1}) + D(a_p) + D(a_{p+1}, a_j) = R(a_i, a_{p-1}) + MTR_{SC}(a_{p-1}) + D(a_p, a_j) \leq R(a_i, a_{p-1}) + TR_{SC}(U(a_i, a_j), a_{p-1}) + D(a_p, a_j) = u(a_i, a_j)$. Hence, we have (22) below.

$$R(a_i, a_p) + D(a_{p+1}, a_j) \leq u(a_i, a_j) \qquad (22)$$

According to Definition 2, (22) means that $U(a_i, a_j)$ is still of SC after execution of a_p.

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

3) The proof is similar to 3) of Theorem 1, hence omitted.

Thus, in overall terms, the theorem holds. ▮

Theorem 3. At activity point a_p, if $R(a_p) \leq M(a_p) + MTR_{WC}(a_{p-1})$, then:

1) all previous SC upper bound constraints are now still of SC;
2) all previous WC upper bound constraints are now at least of WC and may be of SC; and
3) if previous WC upper bound constraints are now still of WC, the status of them has been changed closer to SC.

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

2) Suppose $U(a_k, a_l)$ is of WC before execution of a_p. According to Definition 2, we have (23) below.

$$R(a_k, a_{p-1}) + M(a_p, a_l) \leq u(a_k, a_l) < R(a_k, a_{p-1}) + D(a_p, a_l) \qquad (23)$$

Together with $R(a_p) \leq MTR_{WC}(a_{p-1}) + M(a_p)$, then $R(a_k, a_p) + M(a_{p+1}, a_l) = R(a_k, a_{p-1}) + R(a_p) + M(a_{p+1}, a_l) \leq R(a_k, a_{p-1}) + MTR_{WC}(a_{p-1}) + M(a_p) + M(a_{p+1}, a_l) = R(a_k, a_{p-1}) + MTR_{WC}(a_{p-1}) + M(a_p, a_l) \leq R(a_k, a_{p-1}) + TR_{WC}(U(a_k, a_l), a_{p-1}) + M(a_p, a_l) = u(a_k, a_l)$. Hence, we have (24) below.

$$R(a_k, a_p) + M(a_{p+1}, a_l) \leq u(a_k, a_l) \qquad (24)$$

In addition, $R(a_k, a_p) + D(a_{p+1}, a_l) = R(a_k, a_{p-1}) + R(a_p) + D(a_{p+1}, a_l) \leq R(a_k, a_{p-1}) + MTR_{WC}(a_{p-1}) + M(a_p) + D(a_{p+1}, a_l) \leq R(a_k, a_{p-1}) + MTR_{WC}(a_{p-1}) + D(a_p) + D(a_{p+1}, a_l) = R(a_k, a_{p-1}) + MTR_{WC}(a_{p-1}) + D(a_p, a_l)$. Hence, we have (25) below.

$$R(a_k, a_p) + D(a_{p+1}, a_l) \leq R(a_k, a_{p-1}) + MTR_{WC}(a_{p-1}) + D(a_p, a_l) \qquad (25)$$

However, (24) and (25) are insufficient to judge whether (26) below holds or not.

$$u(a_k, a_l) < R(a_k, a_p) + D(a_{p+1}, a_l) \qquad (26)$$

In fact, depending on specific $MTR_{WC}(a_{p-1})$, (26) may or may not hold. If (26) holds, then, with (24), we have (27) below.

$$R(a_k, a_p) + M(a_{p+1}, a_l) \leq u(a_k, a_l) < R(a_k, a_p) + D(a_{p+1}, a_l) \qquad (27)$$

According to Definition 2, (27) means that $U(a_k, a_l)$ is of WC. If (26) does not hold, according to Definition 2, $U(a_k, a_l)$ already switches to SC after execution of a_p.

3) If $R(a_p) \leq MTR_{WC}(a_{p-1}) + M(a_p)$, then we have (28) below.

$$R(a_p) \leq TR_{WC}(U(a_k, a_l), a_{p-1}) + M(a_p) \qquad (28)$$

With (28), then $R(a_k, a_p) + M(a_{p+1}, a_l) = R(a_k, a_{p-1}) + R(a_p) + M(a_{p+1}, a_l) \leq R(a_k, a_{p-1}) + TR_{WC}(U(a_k, a_l), a_{p-1}) + M(a_p) + M(a_{p+1}, a_l) = R(a_k, a_{p-1}) + TR_{WC}(U(a_k, a_l), a_{p-1}) + M(a_p, a_l)$. Therefore, we have (29) below.

$$R(a_k, a_p) + M(a_{p+1}, a_l) \leq R(a_k, a_{p-1}) + M(a_p, a_l) + TR_{WC}(U(a_k, a_l), a_{p-1}) \qquad (29)$$

Based on (29), we have (30) below.

$$u(a_k, a_l) - [R(a_k, a_{p-1}) + M(a_p, a_l) + TR_{WC}(U(a_k, a_l), a_{p-1})] \leq u(a_k, a_l) - [R(a_k, a_p) + M(a_{p+1}, a_l)] \qquad (30)$$

(30) means that after execution of a_p, $U(a_k, a_l)$ is closer to SC than before.

Thus, in overall terms, the theorem holds. ∎

5.2 Checkpoint Selection along Grid Workflow Execution

According to Figure 5 and Section 5.1, at a_p, we can draw the following three conclusions:

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, we borrow some conclusions from [Chen and Yang 2005b] and we need not verify them either. In [Chen and Yang 2005b], we already developed a method to adjust WC upper bound constraints so that they can still be kept as SC. According to Theorem 3, after the execution of a_p, the status of previous WC upper bound constraints has been 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, the previous adjustment can still be carried forward. Hence, we need not do anything further to it. That is to say, we need not verify it.

14

Based on the above three conclusions, we can decide whether we should take a_p as a checkpoint when grid workflow execution arrives at a_p. The decision-making approach is denoted as *CDA* (Checkpoint Decision-making Approach). The judgement process of *CDA* at a_p is described below.

At activity a_p, if $R(a_p)>D(a_p)+MTR_{SC}(a_{p-1})$, we select 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 select 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 select a_p as a checkpoint.

We now illustrate the correctness of *CDA*. We continue to consider the two execution instances used in Section 4.1. For the execution instance where U_1 and U_2 are of SC at a_{k8}, we suppose $R(a_{k9})=14$. Then at a_{k9}, we have (31) below.

$$D(a_{k9})+MTR_{SC}(a_{k8})=7+6=13 \tag{31}$$

Apparently, we have (32) below.

$$R(a_{k9})>D(a_{k9})+MTR_{SC}(a_{k8}) \tag{32}$$

According to *CDA*, this inequation means that a_{k9} is selected as a checkpoint. In fact, we have (33) below.

$$R(a_{k8})+R(a_{k9})+D(a_{k10})+D(a_{k11})+D(a_{k12})=10+14+10+9+8=51 \tag{33}$$

Since $u(U_2)=50$, we then have (34) below.

$$R(a_{k8})+R(a_{k9})+D(a_{k10})+D(a_{k11})+D(a_{k12})>u(U_2) \tag{34}$$

According to Definition 2, the inequation means that U_2 is not of SC, i.e., it is violated. Hence, we do need to select a_{k9} as a checkpoint so that we can identify the violation. With another execution instance where U_1 and U_2 are of WC at a_{k8}, we can similarly demonstrate: if $M(a_p)+MTR_{WC}(a_{p-1})<R(a_p)\leq D(a_p)+MTR_{SC}(a_{p-1})$, we do need to select a_p as a checkpoint; and if $R(a_p)\leq M(a_p)+MTR_{WC}(a_{p-1})$, we need not take a_p as a checkpoint. In overall term, *CDA* is correct.

Combining *CDA* with DOMTR from Section 4.3 and Appendix A, we can derive a novel checkpoint selection strategy that adaptively selects not only necessary but also sufficient checkpoints dynamically along grid workflow execution. We denote the strategy as CSS_{MTR} (CSS_{MTR}: Minimum Time Redundancy based Checkpoint Selection Strategy). The overall selection process of CSS_{MTR} is:

Along grid workflow execution, CSS_{MTR} calls DOMTR to compute minimum SC and WC time redundancy of each activity. Then when grid workflow execution arrives at an activity, say a_p, CSS_{MTR} calls CDA to decide whether a_p should be taken as a checkpoint.

The overall structure of CSS_{MTR} is depicted in Algorithm 1 below. Algorithm 1 is based on DOMTR. Since DOMTR is already described in detail in Appendix A, we do not repeat the details of DOMTR here. This means DOMTR is needed together to understand Algorithm 1.

Input	Maximum, minimum and mean durations of all activities; all SC and WC upper bound constraints.
Output	*True* or *False* as an appropriate checkpoint.
Step 1	At run-time instantiation stage, conduct DOMTR (refer to Appendix A) to set up some initial values. Initially, there are no any predefined checkpoints.
1.1. Execute Steps 1 and 2 of DOMTR to obtain all $SMTD_{SC-init}$ and $SMTD_{WC-init}$.	
1.2. Execute Step 3 of DOMTR to obtain $EMTD_{SC-init}$ and $EMTD_{WC-init}$ for every end activity of each SC or WC upper bound constraint.	

1.3.	Execute Step 4 of DOMTR to set the biggest possible float number of the system to MTR_{SC} and MTR_{WC} of each activity that is not covered by any SC or WC upper bound constraints.
Step 2	At run-time execution stage, conduct DOMTR (refer to Appendix A) to obtain $MTR_{SC}(a_{p-1})$ and $MTR_{WC}(a_{p-1})$ when grid workflow execution arrives at a_{p-1}.
2.1.	If a_{p-1} is a start activity of some SC and/or WC upper bound constraints, execute Step 5 of DOMTR including Steps 5.1, 5.2 and 5.3 to obtain $MTR_{SC}(a_{p-1})$ and $MTR_{WC}(a_{p-1})$.
2.2.	If a_{p-1} is an intermediate activity of some SC and/or WC upper bound constraints, execute Step 6 of DOMTR to obtain $MTR_{SC}(a_{p-1})$ and $MTR_{WC}(a_{p-1})$.
2.3.	If a_{p-1} is an end activity of some SC and/or WC upper bound constraints, execute Step 7 of DOMTR including Steps 7.1, 7.2, 7.2.1, 7.2.2 to obtain $MTR_{SC}(a_{p-1})$ and $MTR_{WC}(a_{p-1})$.
2.4.	If a_{p-1} is not covered by any SC or WC upper bound constraints, execute Step 8 of DOMTR to obtain $MTR_{SC}(a_{p-1})$ and $MTR_{WC}(a_{p-1})$.
Step 3	At run-time execution stage, call $CDA(a_p)$ to decide whether a_p should be selected as an appropriate checkpoint when grid workflow execution arrives at a_p.
3.1.	Call CDA to compare $D(a_p) + MTR_{SC}(a_{p-1})$ and $M(a_p) + MTR_{WC}(a_{p-1})$ with $R(a_p)$ so that we can decide whether a_p should be selected as an appropriate checkpoint.

Algorithm 1. Checkpoint selection process of CSS_{MTR}

5.3 Necessity and Sufficiency of CSS_{MTR}

We now further prove that checkpoints selected by CSS_{MTR} adaptively along grid workflow execution are both necessary and sufficient for temporal verification.

Theorem 4 (*Necessity*). Along grid workflow execution, all checkpoints selected by CSS_{MTR} are necessary, i.e. there are no any unnecessary checkpoints.

Proof: According to Figure 5 and the three conclusions in Section 5.2, we can see that once we take an activity, say a_p, as a checkpoint, there must be at least one WC or SC upper bound constraint which will be violated. It is exactly the one whose minimum WC or SC time redundancy at a_{p-1} is $MTR_{WC}(a_{p-1})$ or $MTR_{SC}(a_{p-1})$. That is to say, selecting a_p as a checkpoint is necessary.

Thus, the theorem holds. ▌

Theorem 5 (*Sufficiency*). Along grid workflow execution, the checkpoints selected by CSS_{MTR} are sufficient, i.e. there are no any omitted checkpoints.

Proof: With CSS_{MTR}, at a_p, we consider whether we should take it as a checkpoint only if $M(a_p) + MTR_{WC}(a_{p-1}) < R(a_p)$. In fact, according to the discussion in Sections 5.1 and 5.2, we need not take a_p as a checkpoint if $M(a_p) + MTR_{WC}(a_{p-1}) \geq R(a_p)$. Therefore, the checkpoints selected by CSS_{MTR} are sufficient, i.e. none are omitted.

Thus, the theorem holds. ▌

6. SIMULATION AND COMPARISON

In Section 3, we have analysed the problem of $CSS_1 \sim CSS_7$ on checkpoint selection. That is: CSS_1 and CSS_2 may select some unnecessary checkpoints although they do not omit any necessary ones; CSS_3 and CSS_4 may select some unnecessary checkpoints and omit

some necessary ones; and CSS_5, CSS_6 and CSS_7 may select some unnecessary checkpoints without omitting any necessary ones. In Section 5.3, we have rigorously proved the necessity and sufficiency of our strategy CSS_{MTR}. Therefore, CSS_{MTR} is apparently better than $CSS_1 \sim CSS_7$.

We now perform a simulation experiment in our grid workflow management system called SwinDeW-G (**Swin**burne **De**centralised **W**orkflow for **G**rid) [SwinDeW-G 2007; Yan et al. 2006]. In general, on one hand, the simulation experiment simulates the execution of $CSS_1 \sim CSS_7$ & CSS_{MTR} and finds out their respective checkpoints along grid workflow execution. On the other hand, it conducts temporal verification at each activity to check out the real necessary and sufficient checkpoints. Then, by comparing the checkpoints selected by $CSS_1 \sim CSS_7$ and CSS_{MTR} with those by the temporal verification, the simulation experiment demonstrates two results: (1) the necessity and sufficiency of CSS_{MTR} and (2) the significant improvement of CSS_{MTR} on checkpoint selection over $CSS_1 \sim CSS_7$.

In Section 6.1, we describe the simulation environment. We then detail the simulation process of $CSS_1 \sim CSS_7$ and CSS_{MTR} in Section 6.2. In Section 6.3, we depict and analyse the simulation outcomes to demonstrate that the above two results are achieved.

6.1 Simulation Environment

The key component in our simulation environment is SwinDeW-G which is running on a grid testbed [SwinDeW-G 2007]. An overall picture of the testbed is depicted in the bottom plane of Figure 6 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 [CROWN 2006, SwinDeW-G 2007] where CROWN (**C**hina **R**&D Environment **O**ver **W**ide-area **N**etwork) 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).
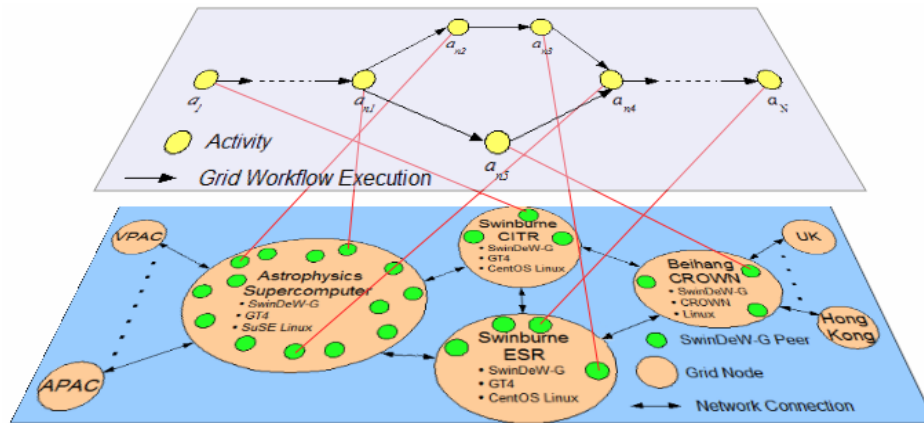

Fig. 6. Overview of the Simulation Environment

17

Currently, SwinDeW-G is deployed at all primary hosting nodes. SwinDeW-G is a peer-to-peer based grid workflow management system [SwinDeW-G 2007]. 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 6, each grid node can have a number of peers. A peer can be simply viewed as a grid service [SwinDeW-G 2007]. In the top plane of Figure 6, we show a sample of how a grid workflow can be executed in the simulation environment.

6.2 Simulation Process

We have simulated $CSS_1 \sim CSS_7$ and CSS_{MTR} on the climate modelling grid workflow execution intensively with different numbers of activities and sub-activities. The simulation process is detailed below.

The whole simulation process consists of two independent sub-processes on the fly with grid workflow execution. One is a checkpoint selection process during which $CSS_1 \sim CSS_7$ and CSS_{MTR} are executed to identify which activities are selected as checkpoints by each of them. That is to say, the checkpoint selection rules of $CSS_1 \sim CSS_7$ described in Section 3 and those of CSS_{MTR} described in Section 5 are performed. The other is a verification process during which all upper bound constraints are verified according to Definition 2 at each activity to check out whether the activity must be selected as a checkpoint. Only those activities where one or more upper bound constraints are violated should be selected as checkpoints so that we can identify and handle the violation in time. Accordingly, the checkpoints identified by the verification process are necessary and sufficient. The two sub-processes are independent of each other and are executed in parallel. Hence, we can compare them with each other in order to derive unnecessary and omitted checkpoints of each strategy.

For example, we consider one of the strategies, say CSS_3. Suppose the grid workflow execution now arrives at activity a_i, say an activity for computing the impact of local thunderstorms on overall climate. After a_i, the factor of local thunderstorms is taken into consideration for the overall climate modelling. Then, at a_i, on one hand, the checkpoint selection process is executed. Namely, the checkpoint selection rules of CSS_3 described in Section 3 are performed to identify whether a_i is selected as a checkpoint by it. This is the checkpoint selection result of CSS_3 at a_i. On the other hand, the verification process is executed. Namely, all upper bound constraints which cover a_i are verified according to Definition 2. If there are any upper bound constraints violated, i.e. not of SC and WC, then a_i must be selected as a checkpoint. Otherwise, it should not be selected as a checkpoint. This is the verification result at a_i and is independent of the checkpoint selection result of CSS_3 at a_i. After this, the two results are compared. There are four possible mapping cases between them.

1) *Case* 1: a_i is selected as a checkpoint by CSS_3, but the verification result indicates it should not be selected as a checkpoint. This means that CSS_3 selected one more unnecessary checkpoint.
2) *Case* 2: a_i is selected as a checkpoint by CSS_3, and the verification result also indicates it must be selected as a checkpoint. This means that CSS_3 did the right thing on checkpoint selection.
3) *Case* 3: a_i is not selected as a checkpoint by CSS_3, but the verification result indicates it must be selected as a checkpoint. This means that CSS_3 omitted one more necessary checkpoint.
4) *Case* 4: a_i is not selected as a checkpoint by CSS_3, and the verification result indicates it should not be selected as a checkpoint either. This means that CSS_3 did the right thing on checkpoint selection.

18

6.3 Simulation Results and Analysis

Based on the above simulation process, we can identify how many unnecessary checkpoints selected by $CSS_1 \sim CSS_7$ and CSS_{MTR}, and how many omitted checkpoints by $CSS_1 \sim CSS_7$ and CSS_{MTR} for each time grid workflow execution. Such results, together with corresponding trajectories, are depicted in Figures 7 and 8 respectively. They change by the number of total grid workflow activities.
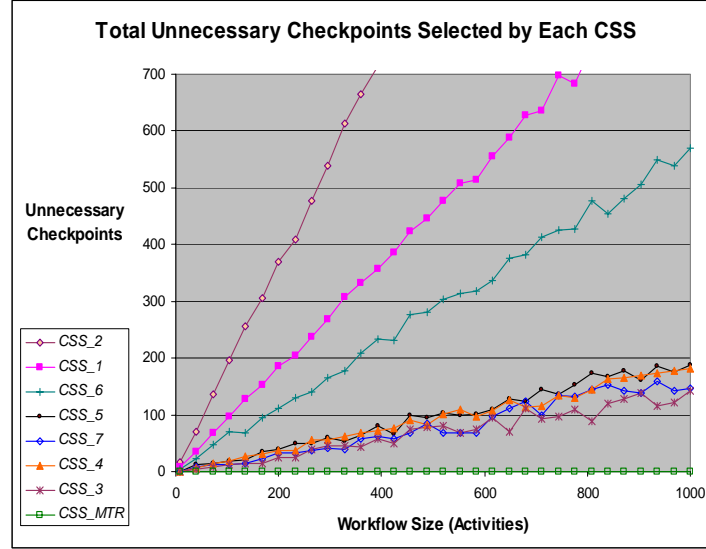

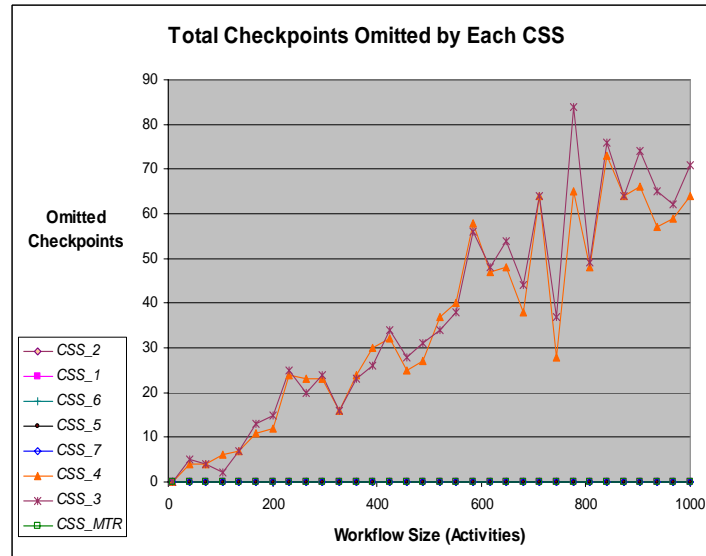Fig. 7. Unnecessary checkpoints selected by each strategy

Fig. 8. Omitted checkpoints by each strategy

19

From Figures 7 and 8, we can see that CSS_{MTR} neither selects any unnecessary checkpoints nor omits any necessary ones. Accordingly, this has further demonstrated the necessity and sufficiency of CSS_{MTR}, i.e. Theorems 4 and 5 derived in Section 5.3.

In addition, from Figure 7, we can see that $CSS_1 \sim CSS_7$ select some unnecessary checkpoints. In particular, when the grid workflow size is getting larger, i.e. there are more stages with more sequential and concurrent activities, the number of unnecessary checkpoints is also getting much bigger. Since real-world grid workflows are normally very complicated and may contain hundreds of thousands of sequential activities [Abramson et al. 2004; Deelman et al. 2003; Simpson et al. 2004], the size of a real-world grid workflow is normally very large indeed. Thus, $CSS_1 \sim CSS_7$ often select a big number of unnecessary checkpoints rather than only a few. As such, comparing with CSS_{MTR} which does not select any unnecessary checkpoints, we can conclude that the improvement of CSS_{MTR} on unnecessary checkpoint selection over $CSS_1 \sim CSS_7$ can be very significant.

From Figure 8, we can see that CSS_3 and CSS_4 omit some necessary checkpoints although CSS_1, CSS_2, CSS_5, CSS_6 and CSS_7 do not omit any. In particular, when the size of a grid workflow is getting larger, i.e. there are more stages with more sequential and concurrent activities, the number of omitted checkpoints is getting much bigger. Similarly to the above analysis of Figure 7, the size of a real-world grid workflow is normally very large indeed. Thus, CSS_3 and CSS_4 often omit a big number of necessary checkpoints rather than only a few. As such, comparing with CSS_{MTR} which does not omit any necessary checkpoints, we can conclude that the improvement of CSS_{MTR} on omitted checkpoint selection over CSS_3 and CSS_4 can also be very significant.

In overall terms, the simulation results have demonstrated the necessity and sufficiency of CSS_{MTR}, and its significant improvement on checkpoint selection over $CSS_1 \sim CSS_7$.

7. CONCLUSIONS AND FUTURE WORK

In grid workflow systems, to verify temporal constraints efficiently at run-time execution stage, checkpoints are often selected so that temporal verification is conducted only at such checkpoints rather than at all activity points. However, this is a complex issue and existing representative checkpoint selection strategies often select some unnecessary checkpoints and omit some necessary ones as they cannot adapt to dynamics and uncertainty of run-time activity completion duration. To overcome such limitations, in this paper, we have proposed a novel checkpoint selection strategy, named CSS_{MTR} (Minimum Time Redundancy based Checkpoint Selection Strategy), which adaptively selects only necessary and sufficient checkpoints along grid workflow execution. Specifically, a new concept of minimum time redundancy has been introduced with a method named DOMTR (Dynamic Obtaining of Minimum Time Redundancy) on how to dynamically obtain minimum time redundancy along grid workflow execution. DOMTR obtains minimum time redundancy on the fly from run-time activity completion duration. Accordingly, minimum time redundancy can adapt to dynamics and uncertainty of run-time activity completion duration. Minimum time redundancy has been used to serve as a key reference parameter for CSS_{MTR} to select checkpoints. Then, relationships between minimum time redundancy and temporal consistency have been investigated in depth. Based on DOMTR and the relationships, CSS_{MTR} has been developed with its necessity and sufficiency rigidly proved. The simulation has further experimentally demonstrated its necessity and sufficiency and its significant improvement on checkpoint selection over other representative strategies.

20

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

ACKNOWLEDGMENTS

REFERENCES

AALST, VAN DER, W.M.P. 1998. The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems and Computers 8(1)*, 21–66.

AALST, VAN DER, W.M.P. 2000. Workflow Verification: Finding Control-Flow Errors using Petri-net based Techniques. In *Proceedings of Business Process Management: Models, Techniques, and Empirical Studies*, Berlin, 2000, Springer-Verlag, LNCS 1806, 161-183.

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)*, 163-182.

ABRAMSON, D., KOMMINENI, J., MCGREGOR, J. L., AND KATZFEY, J. 2004. An Atmospheric Sciences Workflow and Its Implementation with Web Services. In *Proceedings of the 4th International Conference on Computational Science, Part I* , June 2004, Springer Verlag, LNCS 3036, 164-173.

AMIN, K., LASZEWSKI, G. V., HATEGAN, M., ZALUZEC, N. J., HAMPTON, S., AND ROSSI, A. 2004. GridAnt: A Client-controllable Grid Workflow. In *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)*, Hawaii, Jan. 2004, 210-219.

BUYYA, R., ABRAMSON, D., AND VENUGOPAL, S. 2005. The Grid Economy. *Proceedings of the IEEE 93(3)*, 698-714.

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 (CCGrid 2003)*, Tokyo, May 2003, 198-205.

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 (APSEC2004)*, Busan, Korea, Nov./ Dec. 2004, IEEE CS Press, 30-37.

CHEN, J, AND YANG, Y. 2005a. An Activity Completion Duration based Checkpoint Selection Strategy for Dynamic Verification of Fixed-time Constraints in Grid Workflow Systems. In *Proceedings of the 2nd International Conference on Grid Service Engineering and Management (GSEM2005)*, Erfurt, Germany, Sept. 2005, LNI P-69, 296-310.

CHEN, J., AND YANG, Y. 2005b. Multiple Temporal Consistency States for Dynamic Verification of Upper Bound Constraints in Grid Workflow Systems. In *Proceedings of the 1st IEEE International Conference on e-Science and Grid Computing (e-Science2005)*, Melbourne, Australia, Dec. 2005, IEEE CS Press, 124-131.

CHEN, J., AND YANG, Y. 2005c. 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 (APSEC2005)*, Taipei, Taiwan, Dec. 2005, IEEE CS Press, 299-306.

CHEN, J., AND YANG, Y. 2006. Selecting Necessary and Sufficient Checkpoints for Dynamic Verification of Fixed-time Constraints in Grid Workflow Systems. In *Proceedings of the 4th International Conference on Business Process Management (BPM2006)*, Vienna, Austria, Sept. 2006, Springer-Verlag, LNCS 4102, 445-450.

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, 2006. CROWN portal, http://www.crown.org.cn/en/, accessed on Feb. 1, 2007.

CYBOK, D. 2006. A Grid Workflow Infrastructure. *Concurrency and Computation: Practice and Experience, Special Issue on Workflow in Grid Systems 18(10)*, 1243-1254.

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)*, 9-23.

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 (CAiSE'99)*, June 1999, Springer Verlag, LNCS 1626, 286-300.

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, Springer Verlag, LNCS 3038, 42-49.

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 (GGF5)*, Edinburgh, Scotland, July 2002.

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

HAN, Y., SHETH, A., AND BUSSLER, C. 1998. A Taxonomy of Adaptive Workflow Management. In *Proceedings of Workshop on Towards Adaptive Workflow Systems, in conjunction with the 1998 ACM Conference on Computer Supported Cooperative Work*, Seattle, Washington, USA, Nov. 1998.

HUANG, Y. 2003. JISGA: A JINI-BASED Service-Oriented Grid Architecture. *The International Journal of High Performance Computing Applications 17(3)*, 317-327.

KLINGEMANN, J., WÄSCH, J., ABERER, K. 1999a. Deriving Service Models in Cross-Organizational Workflows. In *Proceedings of the 9th International Workshop on Research Issues on Data Engineering Information Technology for Virtual Enterprises (RIDE-VE'99)*, Sydney, Australia, March 1999, 100-107.

KLINGEMANN, J., WÄSCH, J., ABERER, K. 1999b. Adaptive Outsourcing in Cross-Organizational Workflows. In *Proceedings of the 11th Conference on Advanced Information Systems Engineering (Caise'99)*, Heidelberg, Germany, June 1999, LNCS 1626, 417-421.

KRISHNAN, S., WAGSTROM, P., AND LASZEWSKI, G. V. 2002. GSFL: A Workflow Framework for Grid Services. *Technical Report*, Argonne National Laboratory, Argonne, U.S.A., 2002, http://www-unix.globus.org/cog/papers/gsfl-paper.pdf, accessed on Feb. 1, 2007.

LI, H., YANG, Y., AND CHEN, T.Y. 2004. Resource Constraints Analysis of Workflow Specifications. The Journal of Systems and Software 73(2), 271-285.

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.

LIU, Z. 1998. Performance Analysis of Stochastic Timed Petri Nets Using Linear Programming Approach. *IEEE Transactions on Software Engineering 11(24)*, 1014-1030.

MARINESCU, D. 2002. A Grid Workflow Management Architecture. *Global Grid Forum White Paper*, Aug. 2002, http://www.gridforum.org/mail_archive/gce-wg/2002/Archive/pdf00003.pdf, accessed on Feb. 1, 2007.

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.

REICHERT, M., BAUER, T., DADAM, P. 1999. Enterprise-Wide and Cross-Enterprise Workflow Management: Challenges and Research Issues for Adaptive Workflows. In *Proceedings of Workshop on Enterprise-Wide and Cross-Enterprise Workflow Management*, Oct. 1999, 56-64.

RINDERLE, S., REICHERT, M., AND DADAM, P. 2004. Flexible Support of Team Processes by Adaptive Workflow Systems. *Distributed and Parallel Databases 16(1)*, 91-116.

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

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, 547-554.

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. 2007. System Architecture of SwinDeW-G. http://www.ict.swin.edu.au/personal/jchen/ SwinDeW-G/System_Architecture.pdf, accessed on Feb. 1, 2007.

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

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

YU, J., BUYYA, R. AND THAM, C.K. 2005. QoS-based Scheduling of Workflow Applications on Service Grids. In *Proceedings of 1st IEEE International Conference on e-Science and Grid Computing (e-Science2005)*, Melbourne, Australia, Dec. 2005, IEEE CS Press, 140-147.

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

APPENDIX A. WORKING STEPS OF DOMTR

To get a clear picture about the DOMTR working process intuitively, we depict a sample DOMTR working process in Figure 9 for obtaining MTR_{SC} at run-time execution stage.
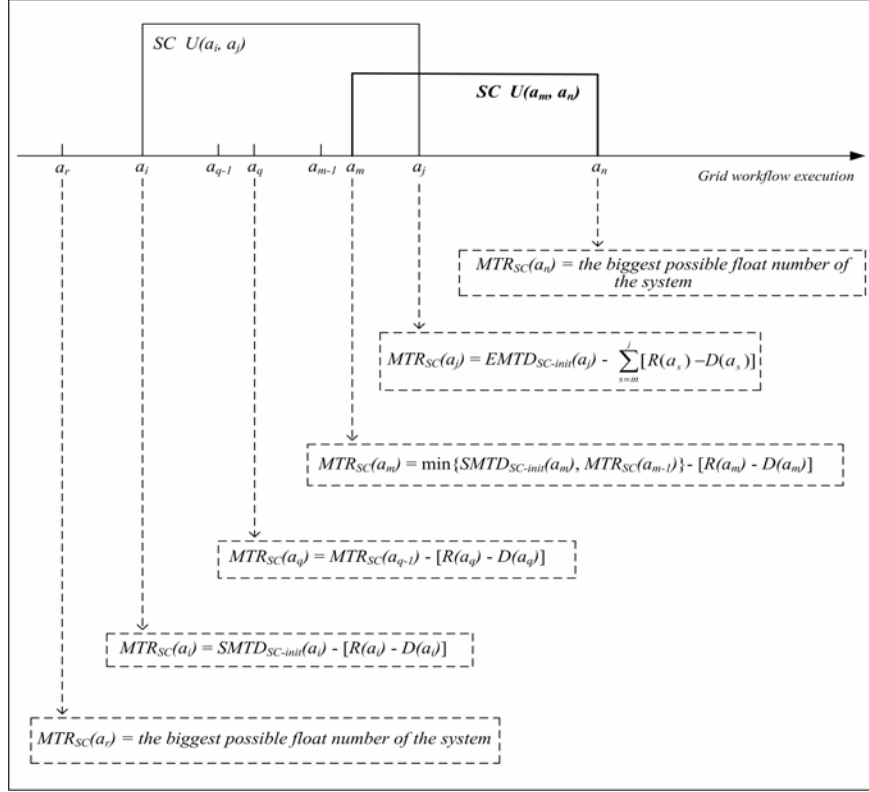
Fig. 9. Sample DOMTR process for obtaining MTR_{SC} at run-time execution stage

Based on Figure 9, we now list the working steps of DOMTR in detail. Due to the space limit, we omit corresponding reasoning about them. Nevertheless, following the steps below, it will not be difficult for readers to understand the rationale as the steps are relatively straightforward.

At run-time instantiation stage (setting up some initial values).

Step 1. During the temporal verification process, for each SC upper bound constraint, say $U(a_i, a_j)$, compute a time difference $u(a_i, a_j) - D(a_i, a_j)$, denoted as SC time difference. For each WC upper bound constraint, say $U(a_k, a_l)$, compute another time difference $u(a_k, a_l) - M(a_k, a_l)$, denoted as WC time difference. All such time differences can be obtained by using corresponding computation results from temporal verification.

Step 2. At every start activity of each SC upper bound constraint, derive minimum SC time difference. For example, at a_i of SC $U(a_i, a_j)$, compare all SC time differences of all SC upper bound constraints that cover a_i to derive the minimum one. Denote it as $SMTD_{SC-init}(a_i)$ (*SMTD*: Minimum Time Difference at Start activity; *init*: initial). At every start activity of each WC upper bound constraint, derive minimum WC time difference. For example, at a_k of WC $U(a_k, a_l)$, compare all WC time differences of all WC upper bound constraints that cover a_k to derive the minimum one. Denote it as $SMTD_{WC-init}(a_k)$.

Step 3. At every end activity of each SC upper bound constraint, derive another minimum SC time difference. But this one is different from the one mentioned in Step 2. For example, at a_j of SC $U(a_i, a_j)$, compare all SC time differences of those SC upper bound constraints which cover a_j, but do not end at a_j. Denote the minimum one as $EMTD_{SC-init}(a_j)$ (*EMTD*: Minimum Time Difference at End activity; *init*: initial). At every

23

end activity of each WC upper bound constraint, derive another minimum WC time difference. For example, at a_l of WC $U(a_k, a_l)$, compare all WC time differences of all WC upper bound constraints which cover a_l, but do not end at a_l. Denote the minimum one as $EMTD_{WC\text{-}init}(a_l)$.

Step 4. For each activity, say a_r, which is not covered by any SC or WC upper bound constraints, set $MTR_{SC}(a_r)$ and $MTR_{WC}(a_r)$ to the biggest possible float number of the system (denoted as BFN) which is far bigger than any $SMTD_{SC\text{-}init}$ and $SMTD_{WC\text{-}init}$.

At run-time execution stage (computing MTR_{SC} and MTR_{WC}).

Step 5. Along grid workflow execution, suppose now the execution arrives at a start activity of some SC and/or WC upper bound constraints, say a_i. There are three situations. The first one is that a_i is the start activity of some SC and WC upper bound constraints. The second one is that a_i is the start activity of some SC upper bound constraints only. The third one is that a_i is the start activity of some WC upper bound constraints only.

Step 5.1. For the first situation, for $MTR_{SC}(a_i)$, if $SMTD_{SC\text{-}init}(a_i) < MTR_{SC}(a_{i-1})$, then, $MTR_{SC}(a_i) = SMTD_{SC\text{-}init}(a_i)-[R(a_i)-D(a_i)]$. Otherwise, $MTR_{SC}(a_i)=MTR_{SC}(a_{i-1})-[R(a_i) - D(a_i)]$. For $MTR_{WC}(a_i)$, if $SMTD_{WC\text{-}init}(a_i) < MTR_{WC}(a_{i-1})$, $MTR_{WC}(a_i) = SMTD_{WC\text{-}init}(a_i) - [R(a_i) - M(a_i)]$. Otherwise, $MTR_{WC}(a_i) = MTR_{WC}(a_{i-1}) - [R(a_i) - M(a_i)]$.

Step 5.2. For the second situation, for $MTR_{SC}(a_i)$, if $SMTD_{SC\text{-}init}(a_i) < MTR_{SC}(a_{i-1})$, then, $MTR_{SC}(a_i) = SMTD_{SC\text{-}init}(a_i) - [R(a_i) - D(a_i)]$. Otherwise, $MTR_{SC}(a_i) = MTR_{SC}(a_{i-1}) - [R(a_i) - D(a_i)]$. For $MTR_{WC}(a_i)$, if $MTR_{WC}(a_{i-1}) = $ BFN, $MTR_{WC}(a_i) = $ BFN. Otherwise, $MTR_{WC}(a_i) = MTR_{WC}(a_{i-1}) - [R(a_i) - M(a_i)]$.

Step 5.3. For the third situation, for $MTR_{SC}(a_i)$, if $MTR_{SC}(a_{i-1}) = $ BFN, $MTR_{SC}(a_i) = $ BFN. Otherwise, $MTR_{SC}(a_i) = MTR_{SC}(a_{i-1}) - [R(a_i) - D(a_i)]$. For $MTR_{WC}(a_i)$, if $SMTD_{WC\text{-}init}(a_i) < MTR_{WC}(a_{i-1})$, $MTR_{WC}(a_i) = SMTD_{WC\text{-}init}(a_i) - [R(a_i) - M(a_i)]$. Otherwise, $MTR_{WC}(a_i) = MTR_{WC}(a_{i-1}) - [R(a_i) - M(a_i)]$.

Step 6. Along grid workflow execution, suppose now the execution arrives at an activity, say a_p. a_p is covered by some SC or WC upper bound constraints, but is neither a start activity nor an end activity of any SC or WC upper bound constraints. After the execution of a_p, $MTR_{SC}(a_p) = MTR_{SC}(a_{p-1}) – [R(a_p) - D(a_p)]$ and $MTR_{WC}(a_p) = MTR_{WC}(a_{p-1}) – [R(a_p) - M(a_p)]$.

Step 7. Along grid workflow execution, we now discuss how to obtain new MTR_{SC} and MTR_{WC} when grid workflow execution arrives at the end activity of some SC and/or WC upper bound constraints. We discuss how to obtain new MTR_{SC} only. For new MTR_{WC}, the corresponding discussion is similar. Suppose now grid workflow execution arrives at the end activity a_j of some SC upper bound constraints. Denote the SC upper bound constraint corresponding to $MTR_{SC}(a_{j-1})$ as $U(MTR_{SC}(a_{j-1}))$.

Step 7.1. If a_j is not the end activity of $U(MTR_{SC}(a_{j-1}))$, obtain $MTR_{SC}(a_j)$ according to Step 6 as $U(MTR_{SC}(a_{j-1}))$ is still functioning.

Step 7.2. If a_j is the end activity of $U(MTR_{SC}(a_{j-1}))$, then, $MTR_{SC}(a_j)$ can also be obtained according to Step 6. However, such $MTR_{SC}(a_j)$ cannot be used further after the execution of a_j because $U(MTR_{SC}(a_{j-1}))$ will be finished. For example, we cannot compute $MTR_{SC}(a_{j+1})$ based on such $MTR_{SC}(a_j)$. Therefore, after the execution of a_j, we need to compute new $MTR_{SC}(a_j)$ to replace such $MTR_{SC}(a_j)$. The new $MTR_{SC}(a_j)$ depends on two situations. The first one is when there are no any other SC upper bound constraints which cover a_j but do not end at a_j. The second one is when there are some other SC upper bound constraints which cover a_j but do not end at a_j.

Step 7.2.1. For the first situation, the new $MTR_{SC}(a_j)$ is set to BFN.

Step 7.2.2. For the second situation, suppose that the upper bound constraint corresponding to $EMTD_{SC\text{-}init}(a_j)$ is $U(a_m, a_n)$ $(m \leq j < n)$. Then, the new $MTR_{SC}(a_j) = EMTD_{SC\text{-}init}(a_j) - \sum_{s=m}^{j} [R(a_s) - D(a_s)]$.

Step 8. When grid workflow execution arrives at an activity which is not covered by any SC or WC upper bound constraints, do nothing and simply keep the initial values set by Step 4.

Step 9. Along grid workflow execution, repeat all or some of Steps 5, 6, 7 and 8 when applicable.