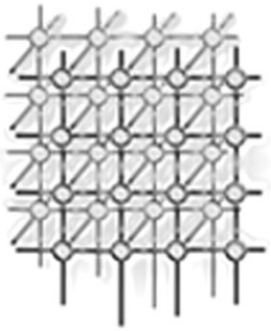
CONCURRENCY AND COMPUTATION: PRACTICE AND EXPERIENCE
Concurrency Computat.: Pract. Exper. 2006; (in press)
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/cpe.1088



Multiple states based temporal consistency for dynamic verification of fixed-time constraints in Grid workflow systems

Jinjun Chen*,† and Yun Yang

Centre for Information Technology Research, Faculty of Information and Communication Technologies, Swinburne University of Technology, P.O. Box 218, Hawthorn, Melbourne, Victoria 3122, Australia

SUMMARY

To verify fixed-time constraints in Grid workflow systems, consistency and inconsistency conditions have been defined in conventional verification work. However, with a view of the run-time uncertainty of activity completion duration, we argue that, although the conventional consistency condition is feasible, the conventional inconsistency condition is too restrictive and covers several different states. These states, which are handled conventionally by the same exception handling, should be handled differently for the purpose of cost saving. Therefore, in this paper, we divide conventional inconsistency into *weak consistency, weak inconsistency and strong inconsistency* and treat conventional consistency as *strong consistency*. Correspondingly, we develop some algorithms on how to verify them. Based on this, for weak consistency we present a method on how to adjust it to strong consistency by using mean activity time redundancy and temporal dependency between fixed-time constraints. For weak inconsistency, we analyse briefly why it can be handled by simpler and more cost-saving exception handling while for strong inconsistency, the conventional exception handling remains deployed. The final quantitative evaluation demonstrates that our research can achieve better cost-effectiveness than the conventional work. Copyright © 2006 John Wiley & Sons, Ltd.

Received 26 April 2006; Accepted 5 May 2006

KEY WORDS: Grid workflow systems; fixed-time constraints; temporal consistency; temporal verification; cost effectiveness

*Correspondence to: Jinjun Chen, Centre for Information Technology Research, Faculty of Information and Communication Technologies, Swinburne University of Technology, P.O. Box 218, Hawthorn, Melbourne, Victoria 3122, Australia.
†E-mail: jchen@ict.swin.edu.au

Contract/grant sponsor: ARC; contract/grant numbers: LP0562500 and DP0663841

1. INTRODUCTION

In Grid architecture, a Grid workflow system is a type of application-level Grid middleware that is supposed to support modelling, redesign and execution of large-scale sophisticated e-science and e-business processes in a variety of complex scientific and business applications such as climate modelling, astrophysics, high-energy physics, structural biology and chemistry, medical surgery, disaster recovery, international banking, insurance, international stock market modelling and control [1–5]. 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 [6,7]. At the build-time stage, complex scientific and business processes are modelled or redesigned as Grid workflow specifications by some Grid workflow definition languages such as Grid Workflow Execution Language (GWEL), Service Workflow Language (SWFL), Abstract Grid Workflow Language (AGWL), or Grid Services Flow Language (GSFL) [6–9]. Conceptually, a Grid workflow contains a large number of activities that are computation, data or transaction intensive, as well as the dependencies between them [6,7]. These activities are implemented and executed by corresponding Grid services [7,10]. The dependencies define activity execution order and form four basic control structures: sequential, parallel, selective and iterative [6,11]. At the run-time instantiation stage, Grid workflow instances are created, and, in particular, Grid services that are specified in the build-time definition documents are discovered by an instantiation Grid service [6,9]. At the run-time execution stage, Grid workflow instances are executed. Execution is coordinated between Grid services by a Grid workflow engine that itself is a high-level Grid service [6,7,9].

In reality, complex scientific and business processes are normally time-constrained [12–14]. Consequently, when corresponding Grid workflow specifications are defined at build-time, fixed-time constraints are often set as well [13,15,16]. A fixed-time constraint at an activity is an absolute time value by which the activity must be completed [13,15,16]. For example, a climate modelling Grid workflow must be completed by the scheduled time [1], say 6:00 p.m. so that the weather forecasting can be broadcast some time later. Here, 6:00 p.m. is a fixed-time constraint. After the fixed-time constraints are set, temporal verification must be conducted at build-time, run-time instantiation and run-time execution stages so that we can identify any temporal violations and consequently can take proper handling actions in time. Particularly at the run-time execution stage, some checkpoints are often selected for conducting temporal verification because it is inefficient to do so at all activity points [16–20]. In addition, at a checkpoint, we only need to consider those fixed-time constraints that cover it. This is because Grid workflow execution at the checkpoint does not affect the consistency of those that do not cover it [17,19,20]. The detailed discussion of checkpoint issues can be found in [16–20]. Here, we simply assume that all checkpoints have already been selected.

According to the literature, [15] uses the modified critical path method (CPM) to calculate temporal constraints, which is one of the very few projects that consider temporal constraint reasoning at both build-time and run-time. A method for dynamic verification of absolute deadline constraints and relative deadline constraints is presented in [16]. A timed workflow process model with consideration of flow time, time difference in a distributed execution environment and consistency checking combining exact run-time duration and estimated build-time duration was proposed. In [21,22], resource constraints and their verification based on some temporal information in workflow systems are discussed. In [23], the author investigates how to develop workflow systems based on component technologies.

DOI: 10.1002/cpe

However, the inconsistency condition defined in the above work is too restrictive although the consistency condition is feasible. The reasons will be detailed in Section 3. To distinguish with the related concepts to be presented later in this paper, we name the consistency and the inconsistency defined in the conventional work as *Conventional Consistency* (CC) and *Conventional Inconsistency* (CI), respectively. In fact, CI involves several different states. Some of them can be handled by potential time saving of succeeding activities without triggering any exception handling to replace the activity of Grid workflow execution, adjust the Grid workflow structure, compensate the completed activities, and so on [24]. Some can be handled by simpler and less costly exception handling. However, in the conventional work, all different situations of CI are generally handled by the same exception handling, which causes some unnecessary extra handling to those that does not need exception handling or can be handled by a simpler one. In real-world Grid workflow systems, every exception handling causes some cost and, in particular, due to the Grid workflow complexity, the handling is normally expensive. Therefore, we should distinguish different situations of CI and discuss corresponding different handlings for them so that we can save the handling cost as much as possible. Hence, in this paper, we divide CI into three states, *Weak Consistency* (WC), *Weak Inconsistency* (WI) and *Strong Inconsistency* (SI), and for integrity we treat CC as *Strong Consistency* (SC). The specific definitions and explanation of the four states will be given in Section 3. Correspondingly, we investigate how to verify them, and develop a method on how to utilize mean activity time redundancy and temporal dependency between fixed-time constraints to adjust WC so that, statistically, it can still be kept like SC without triggering exception handling as in the conventional work. For WI, we discuss why it can rely on simpler and less costly exception handling than SI. The final quantitative evaluation shows that our research can achieve better cost-effectiveness than the conventional work.

The remainder of the paper is organized as follows. Section 2 describes a timed Grid workflow representation. Section 3 details requirements and definitions of SC, WC, WI and SI. Section 4 discusses the verification of SC, WC, WI and SI. Section 5 presents a WC adjustment approach. Section 6 further shows the benefits of our work through a comparison and evaluation. Section 7 concludes our contributions and details future work.

2. TIMED GRID WORKFLOW REPRESENTATION

According to [15,25], 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. Here, we assume that a Grid workflow is well structured. The structure verification is outside of the scope of this paper and can be referred to [26,27]. We borrow some concepts from [15,16,28] such as maximum or minimum duration as a basis to represent time attributes. We denote the ith activity of Grid workflow gw as a_i where a_1 is the first activity of gw. We denote the expected time that the specification of gw will come into effect as $Cie(gw)$. From $Cie(gw)$, the corresponding Grid workflow specification can be used. For each a_i, we denote its maximum duration, minimum duration, run-time start time, run-time end time and run-time completion duration as $D(a_i)$, $d(a_i)$, $S(a_i)$, $E(a_i)$ and $R(a_i)$, respectively. $D(a_i)$ and $d(a_i)$ can be obtained based on the past Grid workflow execution history that is collected by the Grid workflow systems. $R(a_i)$ covers the queuing delay, set-up delay, synchronization delay, network latency, etc. caused at a_i. If there is a fixed-time constraint at a_i, we denote it as $FTC(a_i)$ and its value as $ftv(a_i)$.

DOI: 10.1002/cpe

In addition, we introduce mean duration to each activity. At a_i, we denote it as $M(a_i)$. The activity mean duration means that, statistically, the activity can be completed around its mean duration. According to [29,30], we can apply some stochastic models such as Poisson distribution or exponential distribution to obtain the mean duration for each activity.

If there is a path from a_i to $a_j (i \leq j)$, we denote the maximum duration, minimum duration, mean duration, run-time real completion duration between them as $D(a_i, a_j)$, $d(a_i, a_j)$, $M(a_i, a_j)$ and $R(a_i, a_j)$, respectively [15,16]. For convenience, we consider only one execution path in a Grid workflow without losing generality. As for a selective or parallel structure, each branch is an execution path. Therefore, we can apply the results achieved in this paper to each branch directly. For an iterative structure, from start to end, it is still an execution path. Therefore, we can also apply the results achieved in this paper to it. In overall terms, for a Grid workflow containing many parallel, selective, iterative and/or mixed structures, first, we view 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. Second, for every structure, for each of its branches, we continue to apply the results achieved in this paper. Third, we carry out this recursive process until we complete all branches of all structures. Correspondingly, $D(a_i, a_j)$ is equal to the sum of activity maximum durations between a_i and a_j, and $d(a_i, a_j)$ is equal to the sum of activity minimum durations between a_i and a_j, and $M(a_i, a_j)$ is equal to the sum of activity mean durations between a_i and a_j.

Normally we have $d(a_i) \leq M(a_i) \leq D(a_i)$. There is a difference between $M(a_i)$ and $D(a_i)$. We define it as mean activity time redundancy. Mean activity time redundancy indicates statistically how much redundant time an activity has. It can be used to tolerate certain time deviation of activity execution.

Definition 1. The mean activity time redundancy of a_i is defined as the difference between its maximum duration and mean duration, namely $D(a_i) - M(a_i)$.

3. CC AND CI VERSUS SC, WC, WI AND SI

The definitions of CC and CI in the conventional work can be summarized as follows.

Definition 2. At the build-time stage, $FTC(a_i)$ is said to be of CC if $D(a_1, a_i) \leq ftv(a_i) - Cie(gw)$, and is said to be of CI if $ftv(a_i) - Cie(gw) < D(a_1, a_i)$.

Definition 3. At the run-time instantiation stage, $FTC(a_i)$ is said to be of CC if $D(a_1, a_i) \leq ftv(a_i) - S(a_1)$ and is said to be of CI if $ftv(a_i) - S(a_1) < D(a_1, a_i)$.

Definition 4. At the run-time execution stage, at checkpoint a_p which is either before or at $a_i (p \leq i)$, $FTC(a_i)$ is said to be of CC if $R(a_1, a_p) + D(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1)$, and is said to be of CI if $ftv(a_i) - S(a_1) < R(a_1, a_p) + D(a_{p+1}, a_i)$.

The CC conditions in Definitions 2, 3 and 4 are reasonable because we should try to keep a fixed-time constraint consistent and with such conditions we can reach the maximum largest extent where we can ensure the consistency of a fixed-time constraint. For clarity, in this paper, we denote CC as SC.

Considering the CI, we take Definition 4 as an example to analyse why its condition is too restrictive. The corresponding analysis for Definitions 2 and 3 is similar. In Definition 4, if $ftv(a_i) - S(a_1) < R(a_1, a_p) + D(a_{p+1}, a_i)$, $FTC(a_i)$ will be treated as CI and the exception handling is triggered.

DOI: 10.1002/cpe

However, at the run-time execution stage, the real activity completion duration varies. Therefore, there could be some time redundancy by the succeeding activities after a_i. With this time redundancy, some situations of CI may be able to be adjusted to SC without triggering the exception handling that would cause some extra cost. For those that cannot be adjusted to SC, the specific difference between $ftv(a_i) - S(a_1)$ and $R(a_1, a_p) + D(a_{p+1}, a_i)$ could vary depending on the system load. It may be larger or smaller. For the smaller load, we can trigger the simpler exception handling that may adjust and compensate fewer activities and hence save more cost. For the larger load, we can leave it for the more complicated exception handling as that triggered in the conventional work for all situations of CI.

In summary, we should add more flexibility to CI and divide CI into three different states. The first state can be adjusted to SC with the possible time redundancy without triggering any exception handling. The second state can be treated by simpler and more cost-saving exception handling. The third state can be treated by more complicated exception handling as in the conventional work. Correspondingly, we divide CI into WC, WI and SI. We now give the specific definitions for them and SC.

Definition 5. At the build-time stage, $FTC(a_i)$ is said to be of SC if $D(a_1, a_i) \leq ftv(a_i) - Cie(gw)$, WC if $M(a_1, a_i) \leq ftv(a_i) - Cie(gw) < D(a_1, a_i)$, WI if $d(a_1, a_i) \leq ftv(a_i) - Cie(gw) < M(a_1, a_i)$, and SI if $ftv(a_i) - Cie(gw) < d(a_1, a_i)$.

Definition 6. At the run-time instantiation stage, $FTC(a_i)$ is said to be of SC if $D(a_1, a_i) \leq ftv(a_i) - S(a_1)$, WC if $M(a_1, a_i) \leq ftv(a_i) - S(a_1) < D(a_1, a_i)$, WI if $d(a_1, a_i) \leq ftv(a_i) - S(a_1) < M(a_1, a_i)$, and SI if $ftv(a_i) - S(a_1) < d(a_1, a_i)$.

Definition 7. At the run-time execution stage, at checkpoint a_p, which is either before or at $a_i (p \leq i)$, $FTC(a_i)$ is said to be of SC if $R(a_1, a_p) + D(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1)$, WC if $R(a_1, a_p) + M(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1) < R(a_1, a_p) + D(a_{p+1}, a_i)$, WI if $R(a_1, a_p) + d(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1) < R(a_1, a_p) + M(a_{p+1}, a_i)$, and SI if $ftv(a_i) - S(a_1) < R(a_1, a_p) + d(a_{p+1}, a_i)$.

For clarity, in Figure 1, we further depict the comparison between the definitions of CC and CI and the definitions of SC, WC, WI and SI.

Figure 1 clearly depicts the difference between CC and CI and SC, WC, WI and SI. We now further explain SC, WC, WI and SI. We take the run-time execution stage as an example, i.e. the scenario in Figure 1(c). The corresponding explanation for build-time and run-time instantiation stages is similar. In scenario (c), at a_p, we have the following four situations.

- If $R(a_1, a_p) + D(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1)$, it means $FTC(a_i)$ can be kept if the succeeding activities can be completed by their respective maximum durations. As the maximum duration of an activity is carefully set and mostly should be kept, we define this state as SC.
- If $R(a_1, a_p) + M(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1) < R(a_1, a_p) + D(a_{p+1}, a_i)$, it means if the succeeding activities take their respective maximum durations to complete, $FTC(a_i)$ will be violated. However, if the duration is less than or equal to the mean duration, $FTC(a_i)$ can still be kept. Based on past history, statistically, an activity takes approximately its mean duration to complete, we define this state as WC. It means that with the control of the succeeding activity execution based on the activity mean duration, statistically, $FTC(a_i)$ can still be kept like SC.

DOI: 10.1002/cpe

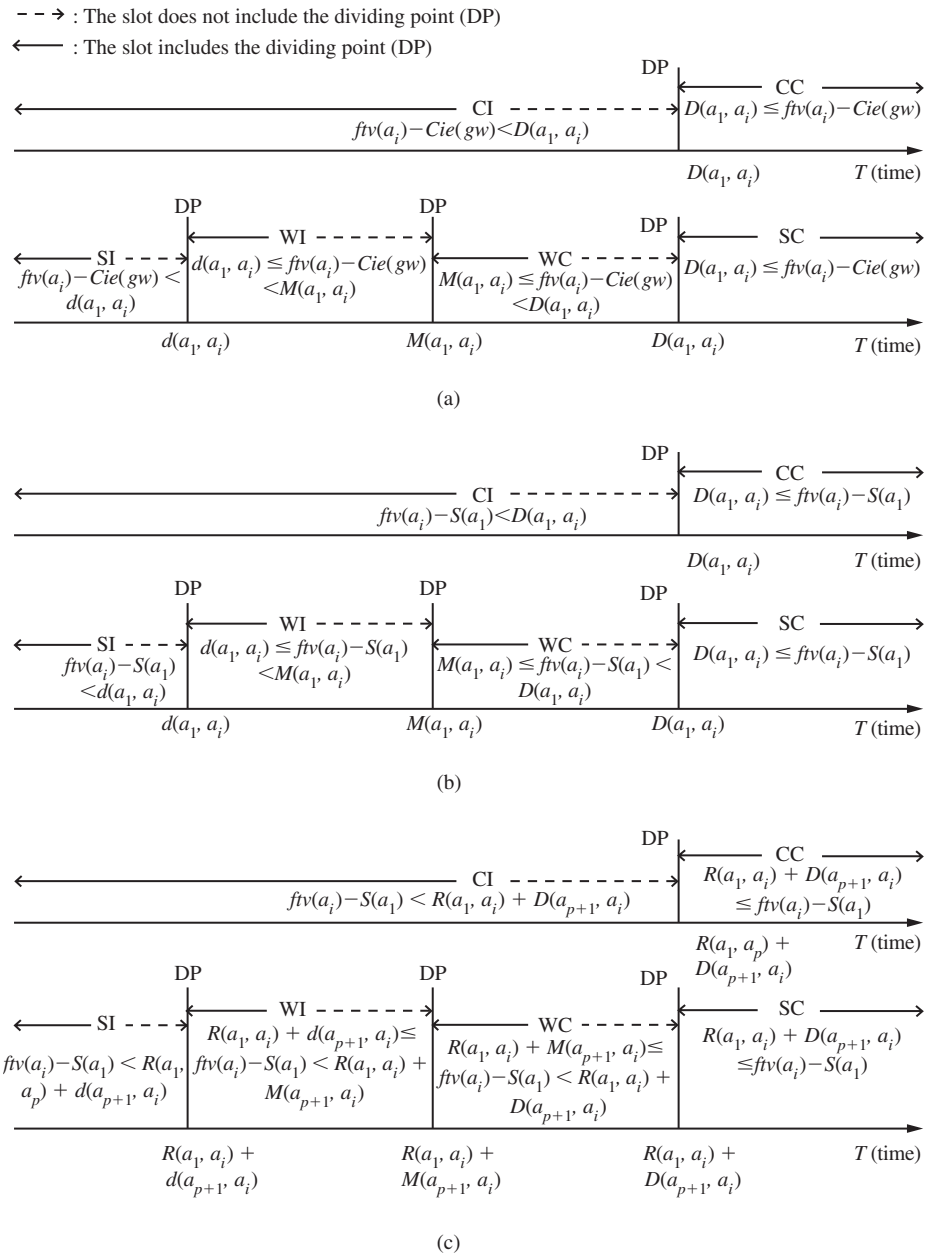


Figure 1. Definitions of CC and CI versus definitions of SC, WC, WI and SI at: (a) build-time stage; (b) run-time instantiation stage; (c) checkpoint a_p at the run-time execution stage.

DOI: 10.1002/cpe

- If $R(a_1, a_p) + d(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1) < R(a_1, a_p) + M(a_{p+1}, a_i)$, it means if the succeeding activities take their respective mean durations or more to complete, $FTC(a_i)$ will be violated. However, if the succeeding activities take about their respective minimum durations to finish, $FTC(a_i)$ can still be kept. According to the setting of the mean and minimum durations, this means that statistically, for most cases, $FTC(a_i)$ is difficult to be kept, and only for fewer cases such as those where all succeeding activities can be completed by their respective minimum durations, $FTC(a_i)$ can be kept. Therefore we define this state as WI.
- If $ftv(a_i) - S(a_1) < R(a_1, a_p) + d(a_{p+1}, a_i)$, it means even if all succeeding activities can be completed by their respective minimum durations, $FTC(a_i)$ still cannot be kept. According to the setting of minimum durations, this means that mostly $FTC(a_i)$ cannot be kept. Therefore, we define this state as SI.

According to the above discussion, definitions and explanation, we need not do anything for SC as in the conventional work. Unlike the conventional work, which treats WC, WI and SI by the same exception handling, we should treat them differently. For WC, we should investigate how to adjust it to SC by using the potential time redundancy. For WI, compared with SI, it has smaller time deficit to recover to SC and there are more cases where it could still be kept like SC. Therefore, we can use simpler and more cost-saving exception handling. However, how simple and cost-saving exception handling should be triggered is beyond the scope of this paper because it is outside the scope of the temporal verification of fixed-time constraints and is often related to some other overall aspects of the whole Grid workflow systems such as quality of service (QoS) [12,14,24]. For SI, we leave it for the more sophisticated and costly exception handling, as in the conventional work [24].

4. TEMPORAL VERIFICATION OF SC, WC, WI AND SI

At the build-time stage, to verify SC, WC, WI and SI, for each fixed-time constraint $FTC(a_i)$, we firstly compute $M(a_1, a_i)$, $d(a_1, a_i)$ and $D(a_1, a_i)$. Then, we compare them with $ftv(a_i) - Cie(gw)$. According to the comparison results and Definition 5, we can judge whether $FTC(a_i)$ is of SC, WC, WI or SI.

At the run-time instantiation stage, some extra activities such as temporal data transfer activities may be added to the Grid workflow specification defined at the build-time stage. In addition, the real start time of a Grid workflow,$S(a_1)$, is instantiated by the Grid workflow engine. $S(a_1)$ may be different from the build-time expected time $Cie(gw)$. Therefore, we need to re-verify the fixed-time constraints. Therefore, we need to re-compute $M(a_1, a_i)$, $d(a_1, a_i)$ and $D(a_1, a_i)$. Then, according to Definition 6, we compare them with $ftv(a_i) - S(a_1)$. According to the comparison results, we can judge whether $FTC(a_i)$ is of SC, WC, WI or SI.

At the run-time execution stage, Grid workflow instances are executed and consequently, we will obtain specific activity completion duration. As the activity completion duration varies depending on the system load, it may impact the consistency of fixed-time constraints [16,17,19,20]. Therefore, we also need to re-verify the fixed-time constraints at selected checkpoints.

Along with Grid workflow execution, at checkpoint a_p, before the execution of a_p, all fixed-time constraints are of either SC or WC because according to Section 3, for WI and SI, we should have already triggered their respective exception handling to change them to SC or WC.

DOI: 10.1002/cpe

In addition, at checkpoint a_p, depending on $R(a_p)$, we may not have to verify all fixed-time constraints that cover a_p. To identify how to verify the fixed-time constraints based on $R(a_p)$, we derive Theorem 1 below.

Theorem 1. *At checkpoint a_p, (1) if $D(a_p) < R(a_p)$, all previous WC fixed-time constraints cannot be of SC. (2) If $R(a_p) \leq D(a_p)$, all previous SC fixed-time constraints are still of SC. (3) If $R(a_p) \leq M(a_p)$, all previous SC fixed-time constraints are still of SC, and all previous WC fixed-time constraints are at least of WC and may be of SC.*

Proof. (1) For a previous WC fixed-time constraint, say $FTC(a_m)(p \leq n)$, according to Definition 7, we have $ftv(a_m) - S(a_1) < R(a_1, a_{p-1}) + D(a_p, a_m)$. If $D(a_p) < R(a_p)$, then, we have $ftv(a_m) - S(a_1) < R(a_1, a_{p-1}) + D(a_p, a_m) = R(a_1, a_{p-1}) + D(a_p) + D(a_{p+1}, a_m) < R(a_1, a_{p-1}) + R(a_p) + D(a_{p+1}, a_m) = R(a_1, a_p) + D(a_{p+1}, a_m)$, namely $ftv(a_m) - S(a_1) < R(a_1, a_p) + D(a_{p+1}, a_m)$. However, according to Definition 7, for $FTC(a_m)$ to be of SC, we must ensure $R(a_1, a_p) + D(a_{p+1}, a_m) \leq ftv(a_m) - S(a_1)$. Therefore, obviously, $FTC(a_m)$ cannot be of SC.

(2) For a previous SC fixed-time constraint, say $FTC(a_s)(p \leq s)$, according to Definition 7, we have $R(a_1, a_{p-1}) + D(a_p, a_s) \leq ftv(a_s) - S(a_1)$. If $R(a_p) \leq D(a_p)$, then we have $R(a_1, a_p) + D(a_{p+1}, a_s) = R(a_1, a_{p-1}) + R(a_p) + D(a_{p+1}, a_s) \leq R(a_1, a_{p-1}) + D(a_p) + D(a_{p+1}, a_s) = R(a_1, a_{p-1}) + D(a_p, a_s) \leq ftv(a_s) - S(a_1)$, namely $R(a_1, a_p) + D(a_{p+1}, a_s) \leq ftv(a_s) - S(a_1)$. According to Definition 7, $FTC(a_s)$ is still of SC after the execution of a_p. Similarly, we can prove it is still of SC if $R(a_p) \leq M(a_p)$.

(3) For a previous WC fixed-time constraint, say $FTC(a_i)(p \leq i)$, according to Definition 7, we have $R(a_1, a_{p-1}) + M(a_p, a_i) \leq ftv(a_i) - S(a_1) < R(a_1, a_{p-1}) + D(a_p, a_i)$. If $R(a_p) \leq M(a_p)$, we have $R(a_1, a_p) + M(a_{p+1}, a_i) = R(a_1, a_{p-1}) + R(a_p) + M(a_{p+1}, a_i) \leq R(a_1, a_{p-1}) + M(a_p) + M(a_{p+1}, a_i) = R(a_1, a_{p-1}) + M(a_p, a_i) \leq ftv(a_i) - S(a_1)$, namely $R(a_1, a_p) + M(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1)$. In addition, we also have: $R(a_1, a_p) + D(a_{p+1}, a_i) = R(a_1, a_{p-1}) + R(a_p) + D(a_p, a_i) \leq R(a_1, a_{p-1}) + M(a_p) + D(a_p, a_i) \leq R(a_1, a_{p-1}) + D(a_p) + D(a_p, a_i) = R(a_1, a_{p-1}) + D(a_p, a_i)$, namely $R(a_1, a_p) + D(a_{p+1}, a_i) \leq R(a_1, a_{p-1}) + D(a_p, a_i)$. However, because of $ftv(a_i) - S(a_1) < R(a_1, a_{p-1}) + D(a_p, a_i)$, we cannot judge whether $ftv(a_i) - S(a_1) < R(a_1, a_p) + D(a_{p+1}, a_i)$ or $R(a_1, a_p) + D(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1)$. In fact, depending on how much $R(a_p)$ is less than $M(a_p)$, each of them may hold. If $ftv(a_i) - S(a_1) < R(a_1, a_p) + D(a_{p+1}, a_i)$, then, with $R(a_1, a_p) + M(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1)$, we have $R(a_1, a_p) + M(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1) < R(a_1, a_p) + D(a_{p+1}, a_i)$. According to Definition 7, $FTC(a_i)$ is of WC. If $R(a_1, a_p) + D(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1)$, according to Definition 7, $FTC(a_i)$ already switches to be of SC. Therefore, $FTC(a_i)$ is at least of WC and may be of SC.

Thus, the theorem holds. □

Based on Theorem 1, at checkpoint a_p, we can derive the relationships between $R(a_p)$ and fixed-time constraint verification. We depict them in Figure 2.

As shown in Figure 2, at checkpoint a_p, we verify the previous SC and WC fixed-time constraints based on the comparison between $R(a_p)$ and $M(a_p)$.

- If $D(a_p) < R(a_p)$, we verify every previous SC fixed-time constraint to see whether it is of SC, WC, WI or SI. We also verify every previous WC fixed-time constraint to judge whether it is of WC, WI and SI. We need not verify its SC because according to Theorem 1, at a_p, if $D(a_p) < R(a_p)$, it is impossible for a previous WC fixed-time constraint to change to be of SC.

DOI: 10.1002/cpe

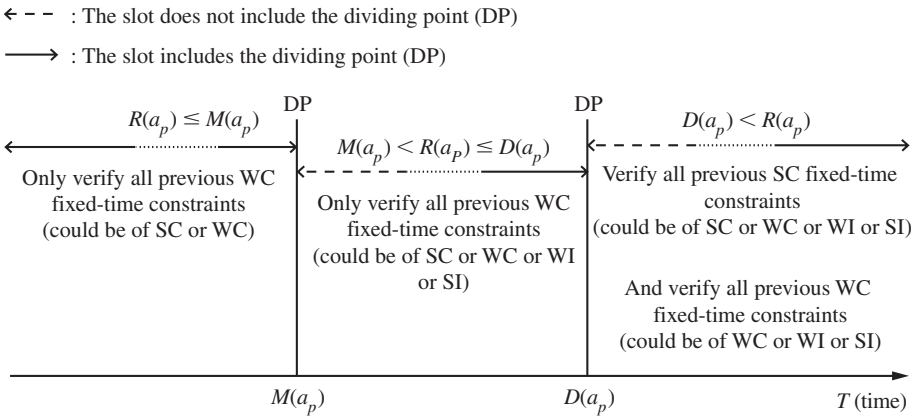


Figure 2. Relationships between completion duration of checkpoint a_p and fixed-time constraint verification.

- If $M(a_p) < R(a_p) \leq D(a_p)$, we verify every previous WC fixed-time constraint to judge whether it is of SC, WC, WI or SI. We need not verify all previous SC fixed-time constraints because according to Theorem 1, they are still of SC.
- If $R(a_p) \leq M(a_p)$, we verify every previous WC fixed-time constraint to judge whether it is of SC or WC. We need not verify its WI or SI because according to Theorem 1, it cannot change to WI or SI. And we need not verify all previous SC fixed-time constraints either because according to Theorem 1, they are still of SC.

After the temporal verification, for WI and SI, as discussed in Section 3, we leave them for their respective exception handling to change to SC or WC. For WC, we discuss how to adjust it to SC in Section 5. For SC, if the fixed-time constraint is previously of SC, we need not do anything. If previously of WC, it means that with the current time redundancy of a_p, the previous WC has already switched to SC. Therefore, we should withdraw the previous adjustment on it that was used to change it to SC.

The detailed fixed-time constraint verification process at checkpoint a_p at the run-time execution stage is presented in Algorithm 1. The corresponding verification processes at the build-time and run-time instantiation stages are simpler and can be derived from Algorithm 1. Hence, we simply omit them.

In Algorithm 1, if $D(a_p) < R(a_p)$, we verify the SC fixed-time constraints and the WC constraints together. Although we can separate them as shown in Figure 2, verifying them together can simplify the algorithm. In addition, according to Theorem 1, if $D(a_p) < R(a_p)$, then at a_p, any previous WC fixed-time constraint cannot be changed to SC. This means that under the condition where $D(a_p) < R(a_p)$, all current SC fixed-time constraints must also be previously of SC. Therefore, we should not withdraw the previous WC adjustment as we still need it. Hence, we do not put any similar withdrawal statement under other conditions.

DOI: 10.1002/cpe

symbol Definitions:
ArraySCWC: an array of all previous SC and WC fixed-time constraints that cover checkpoint a_p;
ArrayWC: an array of all previous WC fixed-time constraints that cover checkpoint a_p;
end Definitions
Input: ArraySCWC, ArrayWC, $S(a_1)$, $R(a_p)$, $M(a_p)$, maximum, minimum and average durations of all activities involved in ArrayFT;
Output: SC, WC, WI and SI report;
If $(D(a_p) < R(a_p))$ **then** //verify all previous SC and WC fixed-time constraints
  while (not end of ArraySCWC) **do**
    Select current fixed-time constraint from ArraySCWC, say $FTC(a_i)(p \leq i)$;
    Compute $R(a_1, a_p)$, $d(a_{p+1}, a_i)$, $D(a_{p+1}, a_i)$ and $M(a_{p+1}, a_i)$;
    if $(R(a_1, a_p) + D(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1))$ **then**
      Output '$FTC(a_i)$ is of SC'
    else if $(R(a_1, a_p) + M(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1) < R(a_1, a_p) + D(a_{p+1}, a_i))$ **then**
      Output '$FTC(a_i)$ is of WC' ; Call WC adjustment // to be discussed in Section 5
    else if $(R(a_1, a_p) + d(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1) < R(a_1, a_p) + M(a_{p+1}, a_i))$ **then**
      Output '$FTC(a_i)$ is of WI' ; Call WI exception handling // beyond the scope of this paper
    else if $(ftv(a_i) - S(a_1) < R(a_1, a_p) + d(a_{p+1}, a_i))$ **then**
      Output '$FTC(a_i)$ is of SI' ; Call SI exception handling // beyond the scope of this paper
    end if
  end while
else if $(M(a_p) < R(a_p) \leq D(a_p))$ **then**
  //only verify all previous WC fixed-time constraints which may be of SC, WC, WI or SI
  while (not end of ArrayWC) **do**
    Select current fixed-time constraint from ArrayWC, say $FTC(a_i)(p \leq i)$;
    Compute $R(a_1, a_p)$, $d(a_{p+1}, a_i)$, $D(a_{p+1}, a_i) and M(a_{p+1}, a_i)$;
    if $(R(a_1, a_p) + D(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1))$ **then**
      Output '$FTC(a_i)$ is of SC'; Withdraw the previous adjustment on $FTC(a_i)$
    else if $(R(a_1, a_p) + M(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1) < R(a_1, a_p) + D(a_{p+1}, a_i))$ **then**
      Output '$FTC(a_i)$ is of WC' ; Call WC adjustment // to be discussed in Section 5
    else if $(R(a_1, a_p) + d(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1) < R(a_1, a_p) + M(a_{p+1}, a_i))$ **then**
      Output '$FTC(a_i)$ is of WI' ; Call WI exception handling // beyond the scope of this paper
    else if $(ftv(a_i) - S(a_1) < R(a_1, a_p) + d(a_{p+1}, a_i))$ **then**
      Output '$FTC(a_i)$ is of SI' ; Call SI exception handling // beyond the scope of this paper
    end if
  end while
else // $R(a_p) \leq M(a_p)$, only verify all previous WC fixed-time constraints which may be of SC or WC
  while (not end of ArrayWC) **do**
    Select current fixed-time constraint from ArrayWC, say $FTC(a_i)(p \leq i)$;
    Compute $R(a_1, a_p)$, $D(a_{p+1}, a_i) and M(a_{p+1}, a_i)$;
    if $(R(a_1, a_p) + D(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1))$ **then**
      Output '$FTC(a_i)$ is of SC'; Withdraw the previous adjustment on $FTC(a_i)$
    else if $(R(a_1, a_p) + M(a_{p+1}, a_i) \leq ftv(a_i) - S(a_1) < R(a_1, a_p) + D(a_{p+1}, a_i))$ **then**
      Output '$FTC(a_i)$ is of WC' ; Call WC adjustment // to be discussed in Section 5
    end if
  end while
end if

Algorithm 1. Verification of SC, WC, WI and SI at checkpoint a_p at run-time execution stage.

DOI: 10.1002/cpe

5. WC ADJUSTMENT

Suppose $FTC(a_i)$ is of WC, to adjust it to SC, at the build-time stage, we have to compensate a time deficit $D(a_1, a_i) - [ftv(a_i) - Cie(gw)]$. At the run-time instantiation stage, the time deficit is $D(a_1, a_i) - [ftv(a_i) - S(a_1)]$. Also at the run-time execution stage, at checkpoint a_p, the time deficit is $R(a_1, a_p) + D(a_{p+1}, a_i) - [ftv(a_i) - S(a_1)]$. Because in real-world Grid workflow systems, there are often many Grid workflow instances in one Grid workflow system, we perform WC adjustment in a statistical way by utilizing the mean activity time redundancy specified in Definition 1. In addition, according to the discussion in [28,31], all fixed-time constraints are nested one after another and hence have temporal dependency on their setting and temporal consistency. By the temporal dependency, we can use the previous WC adjustments to facilitate the current adjustment. Therefore, we would incorporate the temporal dependency into the time deficit allocation to improve its efficiency.

We take the run-time execution stage as an example to discuss the time deficit allocation. The corresponding discussion for the build-time and run-time instantiation stages is similar. At checkpoint a_p, we can only allocate the time deficit to the succeeding activities between a_{p+1} and a_i because all activities before a_p have already been completed and those after a_i have nothing to do with the consistency of $FTC(a_i)$. We denote the time deficit quota to be allocated to a_j between a_{p+1} and a_i by $FTC(a_i)$ as $rdq_i(a_j)$, and the time deficit quota that a_j is holding as $dq(a_j)$. Because statistically, the activity with larger mean activity time redundancy has more time to compensate the time deficit, we allocate the time deficit to a_j based on the proportion of its mean activity time redundancy $D(a_j) - M(a_j)$ out of the overall mean time redundancy of all activities between a_{p+1} and a_i: $\sum_{l=p+1}^{i} [D(a_l) - Mean(a_l)]$. Correspondingly, if there is only one WC fixed-time constraint, say $FTC(a_i)$, we have

$$rdq_i(a_j) = (R(a_1, a_p) + D(a_{p+1}, a_i) - [ftv(a_i) - S(a_1)]) \frac{D(a_j) - M(a_j)}{\sum_{l=p+1}^{i} [D(a_l) - M(a_l)]} (p+1 \le j \le i) \tag{1}$$

To be able to apply formula (1), we must ensure $rdq_i(a_j) \le D(a_j) - M(a_j)$. Otherwise, the available time for a_j to complete is less than its mean duration $M(a_j)$. Therefore, statistically, for most cases, the allocation will probably lead to new WC or WI or SI, which means formula (1) should not be used. Theorem 2 below supports this point.

Theorem 2. *At run-time execution stage, for WC fixed-time constraint $FTC(a_i)$, if we allocate the time deficit to activity a_j between a_{p+1} and a_i according to formula (1), then, we have $rdq_i(a_j) \le D(a_j) - M(a_j)$.*

Proof. To prove $rdq_i(a_j) \le D(a_j) - M(a_j)$, according to (1), we only need to prove $R(a_1, a_p) + D(a_{p+1}, a_i) - [ftv(a_i) - S(a_1)] \le \sum_{l=p+1}^{i} [D(a_l) - M(a_l)] = \sum_{l=p+1}^{i} D(a_l) - \sum_{l=p+1}^{i} M(a_l) = D(a_{p+1}, a_i) - M(a_{p+1}, a_i)$, namely $R(a_1, a_p) + M(a_{p+1}, a_i) \le ftv(a_i) - S(a_1)$. Because $FTC(a_i)$ is of WC, according to Definition 7, we do have $R(a_1, a_p) + M(a_{p+1}, a_i) \le ftv(a_i) - S(a_1)$. Thus, the theorem holds. □

If there are multiple WC fixed-time constraints, because they are nested one after another [28,31], they have some activities in common. For these activities, they will undergo multiple time deficit allocations. Therefore, we have to investigate how to conduct the time deficit allocation for multiple

DOI: 10.1002/cpe

fixed-time constraints together. Based on formula (1), Theorem 2 and the temporal dependency between fixed-time constraints [28,31], we derive a systematic time deficit allocation approach for multiple WC fixed-time constraints. For convenience, we name the approach as fixed-time constraint time deficit allocation (FCTDA). To detail FCTDA, we set a variable, *Maxftv*, that is equal to the maximum of all previously allocated time deficits. We also suppose that the current WC fixed-time constraint of *Maxftv* is $FTC(a_k)$. Correspondingly, *Maxftv* is equal to $R(a_1, a_p) + D(a_{p+1}, a_k) - [ftv(a_k) - S(a_1)]$.

In detail, FCTDA is defined as follows. *At checkpoint a_p for the first WC fixed-time constraint, we use formula (1) to allocate its time deficit and set it to Maxftv. From the second WC fixed-time constraint onwards, for each one, say $FTC(a_i)$, we compare Maxftv with its time deficit, namely $R(a_1, a_p) + D(a_{p+1}, a_i) - [ftv(a_i) - S(a_1)]$. If $R(a_1, a_p) + D(a_{p+1}, a_i) - [ftv(a_i) - S(a_1)] \leq Maxftv$, we directly use the previous time deficit allocations and need not conduct the time deficit allocation for $FTC(a_i)$. Otherwise, first, we compute the difference between Maxftv and the time deficit of $FTC(a_i)$, namely $R(a_1, a_p) + D(a_{p+1}, a_i) - [ftv(a_i) - S(a_1)] - Maxftv$. Second, we allocate the difference to activities between a_{k+1} and a_i $(p < i)$. In particular, for each activity between a_{k+1} and a_i, say a_j, we use formula (2) below to compute $rdq_i(a_j)$. Then, we compare $rdq_i(a_j)$ with $dq(a_j)$. If $dq(a_j)$ is less, we set $rdq_i(a_j)$ to $dq(a_j)$. Otherwise, we need not do anything. Third, we update Maxftv with the time deficit of $FTC(a_i)$, namely $R(a_1, a_p) + D(a_{p+1}, a_i) - [ftv(a_i) - S(a_1)]$.*

$$rdq_i(a_j) = (R(a_1, a_p) + D(a_{p+1}, a_i) - [ftv(a_i) - S(a_1)] - Maxftv) \times \frac{D(a_j) - M(a_j)}{\sum_{l=k+1}^{i} [D(a_l) - M(a_l)]} \quad (p \leq k;\ k+1 \leq j \leq i) \tag{2}$$

To understand FCTDA better and more clearly, we depict its main part in Algorithm 2.

To be able to use FCTDA, we must answer four questions.

(1) Why we need not conduct the time deficit allocation for $FTC(a_i)$ if *Maxftv* is more than or equal to the time deficit of $FTC(a_i)$?
(2) Based on FCTDA, can we ensure $rdq_i(a_j) \leq D(a_j) - M(a_j)$? Otherwise, FCTDA should not be used because it may incur new WC or WI or SI.
(3) Finally, after all time deficit allocations finish, can we ensure $dq(a_j) \leq D(a_j) - M(a_j)$?
(4) After all time deficit allocations finish, statistically, can every WC fixed-time constraint be adjusted to SC?

To answer these questions, we derive Theorems 3–6.

Theorem 3. *At checkpoint a_p, for WC fixed-time constraint $FTC(a_i)$, if $R(a_1, a_p) + D(a_{p+1}, a_i) - [ftv(a_i) - S(a_1)] \leq Maxftv$, the previous WC adjustments are enough for $FTC(a_i)$ to change to SC.*

Proof. To adjust $FTC(a_i)$ to SC, we have to compensate the time deficit $R(a_1, a_p) + D(a_{p+1}, a_i) - [ftv(a_i) - S(a_1)]$. *Maxftv* is the maximum one of the previously allocated time deficits. Because of $R(a_1, a_p) + D(a_{p+1}, a_i) - [ftv(a_i) - S(a_1)] \leq Maxftv$, the current time deficit of $FTC(a_i)$ has already been compensated by the previous maximum time deficit allocation. Therefore, the previous WC adjustments are enough for $FTC(a_i)$ to change to SC. Thus, the theorem holds. □

DOI: 10.1002/cpe

symbol Definitions:
ArrayAJ: an array of all current WC fixed-time constraints which cover checkpoint a_p;
end Definitions
Input: ArrayAJ, $S(a_1)$, $E(a_p)$, maximum and average durations of all activities involved in ArrayAJ;
Output: WC adjustment report;
while (not end of ArrayAJ) **do**
Select current fixed-time constraint from ArrayAJ, say $FTC(a_i)(p \leq i)$;
Compute Deficit$(a_i) = R(a_1, a_p) + D(a_{p+1}, a_i) - [ftv(a_i) - S(a_1)]$;
If ($FTC(a_i)$ is the first WC fixed-time constraint) **then**
Copy all activities between a_{k+1} and a_i to array variable CopyOfActivityA;
While (not end of CopyOfActivityA) **do**
//time deficit allocation for the first WC fixed-time constraint
Select current activity from CopyOfActivityA, say a_j;
Use formula (2) to compute $rdq_i(a_j)$;
$dq(a_j) = rdq_i(a_j))$
end while
$Maxftv = Deficit(a_i)$
else if ($Deficit(a_i) \leq Maxftv$) **then**
Do nothing
else // $Maxftv < Deficit(a_i)$
Copy all activities between a_{k+1} and a_i to array variable CopyOfActivityB;
// Suppose the WC fixed-time constraint of $Maxftv$ is $FTC(a_k)(p \leq k)$
While (not end of CopyOfActivityB) **do** //time deficit allocation for $FTC(a_k)$
Select current activity from CopyOfActivityB, say a_j;
Use formula (2) to compute $rdq_i(a_j)$;
if ($dq(a_j) < rdq_i(a_j)$) **then**
$dq(a_j) = rdq_i(a_j)$;
end if
end while
$Maxftv = Deficit(a_i)$
end if
end while

Algorithm 2. Time deficit allocation at checkpoint a_p at run-time execution stage for multiple WC fixed-time constraints.

Theorem 4. *In FCTDA, after we apply formula (2) to compute $rdq_i(a_j)$, we still have $rdq_i(a_j) \leq D(a_j) - M(a_j)$.*

Proof. To prove Theorem 4, we borrow some relevant conclusions derived in [28,31] about the temporal dependency between the fixed-time constraints. According to [28,31], the temporal dependency means that the fixed-time constraints are dependent on each other in terms of their setting and verification because they are nested one after another. Correspondingly, given two WC fixed-time constraints $FTC(a_k)$ and $FTC(a_i)$ $(k < i)$, we have $M(a_{k+1}, a_i) \leq ftv(a_i) - ftv(a_k)$. The detailed discussion and the deducing process can be referred to [28,31]. To avoid repetition, we do not address them here again.

DOI: 10.1002/cpe

Now, we prove Theorem 4. To prove $rdq_i(a_j) \leq D(a_j) - M(a_j)$, according to formula (2), we only need to prove $R(a_1, a_p) + D(a_{p+1}, a_i) - [ftv(a_i) - S(a_1)] - Maxftv \leq \sum_{l=k+1}^{i} [D(a_l) - M(a_l)]$. Since $Maxftv = R(a_1, a_p) + D(a_{p+1}, a_k) - [ftv(a_k) - S(a_1)]$ and $\sum_{l=k+1}^{i} [D(a_l) - M(a_l)] = \sum_{l=k+1}^{i} D(a_l) - \sum_{l=k+1}^{i} M(a_l)$, we actually only need to prove $\sum_{l=k+1}^{i} M(a_l) \leq [ftv(a_i) - S(a_1)] - [ftv(a_k) - S(a_1)]$, namely $M(a_{k+1}, a_i) \leq ftv(a_i) - ftv(a_k)$. However, in the above paragraph, we know that we do have $M(a_{k+1}, a_i) \leq ftv(a_i) - ftv(a_k)$. Thus, the theorem holds. □

Theorem 5. *At checkpoint a_p, if we use FCTDA to conduct the time deficit allocation for every WC fixed-time constraint that covers a_p, then, finally, we have $dq(a_j) \leq D(a_j) - M(a_j)$.*

Proof. According to FCTDA, $dq(a_j)$ is equal to the maximum deficit quota allocated to a_j by all WC fixed-time constraints. According to Theorems 2 and 4, any deficit quota allocated to a_j is less than or equal to $D(a_j) - M(a_j)$. Therefore, the maximum value must also be less than or equal to $D(a_j) - M(a_j)$, namely $dq(a_j) \leq D(a_j) - M(a_j)$. Thus, the theorem holds. □

Theorem 6. *At checkpoint a_p, after we use FCTDA to finish all time deficit allocations for all WC fixed-time constraints that cover a_p, statistically the final time deficit quotas of activities between a_{p+1} and a_i are enough for all WC fixed-time constraints to change to SC.*

Proof. Considering any one WC fixed-time constraint, say $FTC(a_i)$, for any a_j between a_{p+1} and a_i, after all time deficit allocations finish, according to FCTDA, we have $rdq_i(a_j) \leq dq(a_j)$. This means that at a_j, by taking $dq(a_j)$, $FTC(a_i)$ can have more time to compensate its time deficit than by its own deficit allocation. Because $FTC(a_i)$ can switch to SC even only based on its own deficit allocation, we can say that based on multiple allocations, $FTC(a_i)$ can be easier to switch to SC. Thus, the theorem holds. □

Obviously, Theorems 3–6 answer all the above four questions, respectively. Therefore, we can apply FCTDA to conduct the time deficit allocation so that, statistically, every WC fixed-time constraint can still be kept as SC.

6. COMPARISON AND QUANTITATIVE EVALUATION

In conventional fixed-time constraint verification work, WC, WI and SI are covered by CI and handled by the same exception handling. However, in this paper, by separating them, we can handle them differently. For WC, by deploying FCTDA, statistically, WC can still be kept as SC. Therefore, we need not trigger any exception handling. For WI, according to the discussion in Section 3, we can resort to simpler and more cost-saving exception handling. As, in the real world every exception handling normally causes some cost such as compensating some completed activities [24], clearly our four-state approach can be more cost-effective than the conventional two-state approach.

We further conduct corresponding quantitative analysis so that we can obtain a specific picture of how our four-state approach is more cost-effective than the conventional two-state method. Similarly to Sections 4 and 5, we only consider the run-time execution stage. The corresponding quantitative analysis for the build-time and run-time instantiation stages is similar. For simplicity, we only consider the major exception handling cost, which is spent on activities such as activity compensation [24].

DOI: 10.1002/cpe

We consider a complex climate modelling Grid workflow, gw, which may consist of hundreds of thousands of sub-activities and must be time constrained for the purpose of weather forecasting [1]. We denote the exception handling cost of a_k as $C_k(gw)$, the number of WC as $N(gw)$, the number of WI as $M(gw)$, the number of SI as $L(gw)$. In addition, we use $Q_i(gw)$ to represent the number of activities addressed by the exception handling conducted by the conventional work for the ith WC or WI or SI, and $P_j(gw)$ to represent the number of activities addressed by the exception handling conducted by our work for the jth WI. We denote the total exception handling cost of the conventional work as $PC_{\text{total}}(gw)$, the total exception handling cost of our research as $OC_{\text{total}}(gw)$. According to the discussion in Section 3, $P_i(gw)$ is less than $Q_i(gw)$ and statistically we have

$$PC_{\text{total}}(gw) = \sum_{i=1}^{N(gw)} \sum_{k=1}^{Q_i(gw)} C_k(gw) + \sum_{j=1}^{M(gw)} \sum_{s=1}^{Q_j(gw)} C_s(gw) + \sum_{l=1}^{L(gw)} \sum_{r=1}^{Q_l(gw)} C_r(gw) \tag{3}$$

$$OC_{\text{total}}(gw) = \sum_{j=1}^{M(gw)} \sum_{m=1}^{P_j(gw)} C_m(gw) + \sum_{l=1}^{L(gw)} \sum_{n=1}^{Q_l(gw)} C_n(gw) \tag{4}$$

Intuitively, $OC_{\text{total}}(gw)$ is less than or equal to $PC_{\text{total}}(gw)$. We then investigate to what extent $OC_{\text{total}}(gw)$ is less than or equal to $PC_{\text{total}}(gw)$. We denote the difference between $PC_{\text{total}}(gw)$ and $OC_{\text{total}}(gw)$ as $DIFF_{\text{total}}(gw)$. Then, based on formulas (3) and (4), we have

$$DIFF_{\text{total}}(gw) = \sum_{i=1}^{N(gw)} \sum_{k=1}^{Q_i(gw)} C_k(gw) + \sum_{j=1}^{M(gw)} \sum_{s=1}^{Q_j(gw)} C_s(gw) - \sum_{j=1}^{M(gw)} \sum_{m=1}^{P_j(gw)} C_m(gw) \tag{5}$$

To obtain an overall analysis, we analyse $DIFF_{\text{total}}(gw)$ in a statistical way. Therefore, we replace the corresponding variables in formula (5) with their respective mean values, which can be achieved based on past execution history. We denote the mean values of $C_k(gw)$, $N(gw)$, $M(gw)$, $L(gw)$, $DIFF_{\text{total}}(gw)$ as C, N, M, L, $DIFF_{\text{total}}$, respectively. In addition, according to [29,30], statistically $Q_i(gw)$ and $P_j(gw)$ can be a kind of stochastic distribution. As we discuss $DIFF_{\text{total}}(gw)$ in a statistical way, we take the mean distribution without losing generality as for other stochastic distributions, the final conclusions being similar. Correspondingly, we have $Q_i(gw) = X * A_i$, $P_j(gw) = Y * B_j$ where A_i stands for the number of activities covered by the ith WC. B_j stands for the number of activities covered by the jth WI, and X and Y are mean weights that depend on the mean system load, available Grid services and the mean time deficit ($0 < X \leq 1$, $0 < Y \leq 1$). According to Section 3, Y is less than X. Then, we can derive

$$DIFF_{\text{total}} = \sum_{i=1}^{N} CXA_i + \sum_{j=1}^{M} C(X - Y)B_j \tag{6}$$

We suppose that the first fixed-time constraint covers P activities. All fixed-time constraints are nested one after another [28,31]. For convenience, we assume that the distance between any two adjacent fixed-time constraints be the same, denoted as Q. Because formula (6) has nothing to do with SI, we assume that there is no SI. In addition, according to the discussion of temporal dependency in [28,31], we know that all fixed-time constraints after a WC one will be of WC or even SC, and all fixed-time constraints after a SC one will be of SC where the detailed explanation and the

DOI: 10.1002/cpe

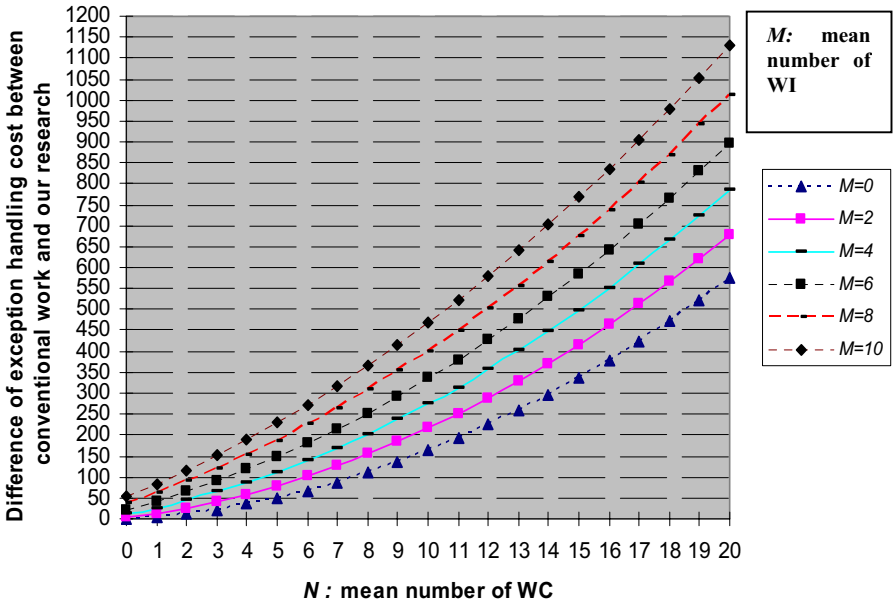


Figure 3. Change trend of exception handling cost difference between conventional work and our research by mean WC number N and mean WI number M.

deducing process can be found in [28,31]. Therefore, we can derive $A_i = [P + (M-1)Q] + iQ$ and $B_j = P + (j-1)Q$. If we apply them to formula (6), then, we have

$$DIFF_{\text{total}} = CXN[P + (M-1)Q] + CXQ\frac{N(N+1)}{2} + C(X-Y)MP + C(X-Y)Q\frac{M(M-1)}{2} \tag{7}$$

We now take a set of specific values to see how formula (7) performs. We suppose $P = 10$, $Q = 5$, $X = 1/2$, $Y = 1/3$, and C is equal to 1 cost unit. We also suppose that statistically N can change from 0 to 20 and M can change from 0 to 10. The selection of these specific values does not affect our analysis because what we want is the trend of how the exception handling cost of our research is less than that of the conventional work with the mean WC number N or the mean WI number M changing. With N and M changing, according to formula (7), we depict corresponding $DIFF_{\text{total}}$ in Figure 3.

According to Figure 3, we can see that with N increasing, $DIFF_{\text{total}}$ is increasing. This means that the more WC, the more exception handling cost our research can save. We can also see that with M increasing, $DIFF_{\text{total}}$ is also increasing. This means that the more WI, the more exception handling cost our research can save. In overall terms, the larger the number of WC or WI, the more costs our approach can save. Therefore, statistically, our research can achieve better cost-effectiveness than the conventional work.

DOI: 10.1002/cpe

7. CONCLUSIONS AND FUTURE WORK

In this paper, based on the analysis of CC (conventional consistency) and CI (conventional inconsistency) conditions defined in the conventional work, and the run-time uncertainty of activity completion duration in Grid workflow systems, we have introduced four states to a fixed-time constraint. They are SC (strong consistency), WC (weak consistency), WI (weak inconsistency) and SI (strong inconsistency). Correspondingly, we have developed their verification methods and algorithms. Furthermore, for WC, we have developed a method for how to utilize mean activity time redundancy and temporal dependency between fixed-time constraints to adjust it so that statistically, it can still be kept as SC. By this method, we need not trigger any exception handling to handle WC as in the conventional work. For WI, we have analysed briefly the reason why it can be treated by simpler exception handling than that used by the conventional work while for SI, the conventional exception handling may still be used. Finally, we have conducted a quantitative evaluation that has demonstrated that our four-state approach can achieve better cost-effectiveness than the conventional two-state method. Because in Grid workflow systems, activity completion duration is highly uncertain and normally every exception handling causes some handling cost, by deploying our four-state approach and corresponding results, we can make the temporal verification more applicable to dynamic Grid workflow systems.

With these contributions, we can further investigate the exception handling for WI with the consideration of some relevant overall aspects of Grid workflow systems such as QoS.

ACKNOWLEDGEMENTS

We thank Professor Hai Zhuge for organizing this special issue. We are also grateful for the constructive comments of the reviewers.

REFERENCES

1. Abramson D, Kommineni J, McGregor JL, Katzfey J. An atmospheric sciences workflow and its implementation with Web services. *Proceedings of the 4th International Conference on Computational Science, Part I*, Krakow, Poland (*Lecture Notes in Computer Science*, vol. 3036). Springer: Berlin, 2004; 164–173.
2. Amin K, Laszewski GV, Hategan M, Zaluzec NJ, Hampton S, Rossi A. GridAnt: A client-controllable Grid workflow. *Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04)*, Hawaii, HI, January 2004. IEEE Computer Society Press: Los Alamitos, CA, 2004; 210–219.
3. Cao J, Jarvis SA, Saini S, Nudd GR. GridFlow: Workflow management for Grid computing. *Proceedings of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, Tokyo, Japan, May 2003. IEEE Computer Society Press: Los Alamitos, CA, 2003; 198–205.
4. Foster I, Kesselman C, Nick J, Tuecke S. The physiology of the Grid: An open Grid services architecture for distributed systems integration, Globus Project, 2002. http://www.globus.org/research/papers/ogsa.pdf [10 March 2006].
5. Simpson DR, Kelly N, Jithesh PV, Donachy P, Harmer TJ, Perrott RH, Johnston J, Kerr P, McCurley M, McKee S. GeneGrid: A practical workflow implementation for a Grid based virtual bioinformatics laboratory. *Proceedings of the U.K. e-Science All Hands Meeting 2004 (AHM04)*, Nottingham, U.K., September 2004. EPSRC: Swindon, 2004; 547–554.
6. Cybok D. A Grid workflow infrastructure. *Concurrency and Computation: Practice and Experience (Workflow in Grid Systems Special Issue)* 2006; **18**(10):1243–1254. DOI: 10.1002/cpe998.
7. Huang Y. JISGA: A JINI-based service-oriented Grid architecture. *The International Journal of High Performance Computing Applications* 2003; **17**(3):317–327.

8. Fahringer T, Pllana S, Villazon A. A-GWL: Abstract Grid workflow language. *Proceedings of the 4th International Conference on Computational Science, Part III*, Krakow, Poland (*Lecture Notes in Computer Science*, vol. 3038). Springer: Berlin, 2004; 42–49.
9. Krishnan S, Wagstrom P, Laszewski GV. GSFL: A workflow framework for Grid services. *Technical Report*, Argonne National Laboratory, Argonne, IL, 2002. Available at: http://www-unix.globus.org/cog/papers/gsfl-paper.pdf [10 March 2006].
10. Deelman E, Blythe J, Gil Y, Kesselman C, Mehta G, Vahi K. Mapping abstract complex workflows onto Grid environments. *Journal of Grid Computing* 2003; **1**(1):9–23.
11. Zhuge H. Workflow-based cognitive flow management for distributed team cooperation. *Information and Management* 2003; **40**(5):419–429.
12. Brandic I, Benkner S, Engelbrecht G, Schmidt R. Towards quality of service support for Grid workflows. *Proceedings of the European Grid Conference 2005 (EGC2005)*, Amsterdam, The Netherlands (*Lecture Notes in Computer Science*, vol. 3470). Springer: Berlin, 2005; 661–670.
13. Buyya R, Abramson D, Venugopal S. The Grid Economy. *Proceedings of the IEEE* 2005; **93**(3):698–714.
14. Yu J, Buyya R, Tham CK. Cost-based scheduling of scientific workflow applications on utility Grids. *Proceedings of the 1st IEEE International Conference on e-Science and Grid Computing (e-Science2005)*, Melbourne, Australia, December 2005. IEEE Computer Society Press: Los Alamitos, CA, 2005; 140–147.
15. Eder J, Panagos E, Rabinovich M. Time constraints in workflow systems. *Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE'99)* (*Lecture Notes in Computer Science*, vol. 1626). Springer: Berlin, 1999; 286–300.
16. Marjanovic O, Orlowska ME. On modelling and verification of temporal constraints in production workflows. *Knowledge and Information Systems* 1999; **1**(2):157–192.
17. Chen J, Yang Y. A minimum proportional time redundancy based checkpoint selection strategy for dynamic verification of fixed-time constraints in Grid workflow systems. *Proceedings of the 12th Asia Pacific Software Engineering Conference (APSEC2005)*, Taiwan, December 2005. IEEE Computer Society Press: Los Alamitos, CA, 2005; 299–306.
18. Chen J, Yang Y, Chen TY. Dynamic verification of temporal constraints on-the-fly for workflow systems. *Proceeding of the 11th Asia-Pacific Software Engineering Conference (APSEC2004)*, Busan, Korea, November/December 2004. IEEE Computer Society Press: Los Alamitos, CA, 2004; 30–37.
19. Chen J, Yang Y. An activity completion duration based checkpoint selection strategy for dynamic verification of fixed-time constraints in Grid workflow systems. *Proceedings of the 2nd International Conference on Grid Service Engineering and Management (GSEM2005)*, Erfurt, Germany (*Lecture Notes in Informatics*, vol. P-69). Springer: Berlin, 2005; 296–310.
20. Zhuge H, Cheung T, Pung H. A timed workflow process model. *The Journal of Systems and Software* 2001; **55**(3):231–243.
21. Li H, Yang Y, Chen TY. Resource constraints analysis of workflow specifications. *The Journal of Systems and Software* 2004; **73**(2):271–285.
22. Li H, Yang Y. Dynamic checking of temporal constraints for concurrent workflows. *Electronic Commerce Research and Applications* 2005; **4**(2):124–142.
23. Zhuge H. Component-based workflow systems development. *Decision Support Systems* 2003; **35**(4):517–536.
24. Hagen C, Alonso G. Exception handling in workflow management systems. *IEEE Transactions on Software Engineering* 2000; **26**(10):943–958.
25. Chinn S, Madey G. Temporal representation and reasoning for workflow in engineering design change review. *IEEE Transactions on Engineering Management* 2000; **47**(4):485–492.
26. Van der Aalst WMP. Workflow verification: Finding control-flow errors using Petri-net based techniques. *Proceedings of the International Conference on Business Process Management: Models, Techniques, and Empirical Studies* (*Lecture Notes in Computer Science*, vol. 1806). Springer: Berlin, 2000; 161–183.
27. Rinderle S, Reichert M, Dadam P. On dealing with structural conflicts between process type and instance changes. *Proceedings of the 2nd International Conference on Business Process Management*, Potsdam, Germany (*Lecture Notes in Computer Science*, vol. 3080). Springer: Berlin, 2004; 274–289.
28. Chen J, Yang Y. Temporal dependency for dynamic verification of temporal constraints in workflow systems. *Proceedings of the 3rd International Conference on Grid and Cooperative Computing*, Wuhan, China (*Lecture Notes in Computer Science*, vol. 3251). Springer: Berlin, 2004; 1005–1008.
29. Ha B, Bae J, Kang S. Workload balancing on agents for business process efficiency based on stochastic model. *Proceedings of the 2nd International Conference on Business Process Management*, Potsdam, Germany (*Lecture Notes in Computer Science*, vol. 3080). Springer: Berlin, 2004; 195–210.
30. Liu Z. Performance analysis of stochastic timed petri nets using linear programming approach. *IEEE Transactions on Software Engineering* 1998; **11**(24):1014–1030.
31. Chen J, Yang Y. Temporal dependency for dynamic verification of fixed-date constraints in Grid workflow systems. *Proceedings of the 7th Asia Pacific Web Conference*, Shanghai, China (*Lecture Notes in Computer Science*, vol. 3399). Springer: Berlin, 2005; 820–831.

DOI: 10.1002/cpe