Multiple Temporal Consistency States for Dynamical Verification of Upper Bound Constraints in Grid Workflow Systems

Jinjun Chen, Yun Yang

Faculty of Information and Communication Technologies
Swinburne University of Technology
PO Box 218, Hawthorn, Melbourne, Australia 3122
{jchen, yyang}@ict.swin.edu.au

Abstract

Conventional upper bound constraint verification in grid workflow systems is based on the key assumption that an upper bound constraint only has two states: consistency and inconsistency. However, due to complexity of grid workflows and dynamic availability of participating grid services, this assumption is too restrictive as there may be some intermediate states. Therefore, in this paper, we introduce four states for an upper bound constraint. Namely, we treat conventional consistency as strong consistency and divide conventional inconsistency into weak consistency, weak inconsistency and strong inconsistency. Correspondingly, we develop their verification methods. For weak consistency, we present some algorithms on how to adjust it to strong consistency without triggering exception handling as in conventional work. For weak inconsistency, we analyse why it can rely on less costly exception handling than conventional work. The final evaluation demonstrates that our four-state approach can achieve better cost-effectiveness than the conventional two-state approach.

1. Introduction

In the grid architecture, a grid workflow system is a type of high-level grid middleware which is supposed to support modelling, redesign and execution of large-scale sophisticated e-science processes in many complex scientific applications such as climate modelling, astrophysics or high energy physics [1, 2, 4, 15, 22]. Generally speaking, the whole working process of a grid workflow system can be divided into three stages: build-time, run-time instantiation and run-time execution [4, 17, 18]. At the build-time stage, complex scientific or business processes are modelled or redesigned as grid workflow specifications by some grid workflow definition languages such as Grid Services Flow Language (GSFL), Abstract Grid Workflow Language (AGWL), or Grid Workflow Execution Language (GWEL) [4, 14, 18, 24]. According to [4, 17], conceptually, a grid workflow contains a lot of computation, data or transaction intensive activities, and dependencies between them. These activities are implemented and executed by corresponding grid services [4, 9, 18]. The dependencies define activity execution orders and form four basic control structures: sequential, parallel, selective and iterative [4, 9, 18]. At the run-time instantiation stage, grid workflow instances are created, and especially grid services which are specified in build-time definition documents are discovered by an instantiation service that is a high-level grid service [4, 9, 18]. 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 too [4, 7, 9, 17, 18].

To control temporal correctness of grid workflow execution, upper bound constraints are often set at build-time [5, 13, 19, 21]. An upper bound constraint between two activities is a relative time value which the duration between the two activities must be less than or equal to. Temporal verification is conducted to check if all upper bound constraints are consistent, and can be applied to the above three stages. And at the run-time execution stage, some checkpoints are often selected for conducting temporal verification because it is not efficient to do so at all activity points [6, 21, 25]. The detailed discussion about checkpoint selection is outside the scope of this paper and can be found in some other references such as [6, 8, 10, 21, 25]. Here, we simply assume that all checkpoints have already been selected.

Some related work has been done and will be detailed in Section 2. However, they are based on the key assumption that an upper bound constraint only has two states: consistency and inconsistency. Due to complexity of grid workflows and dynamic availability of participating grid services, this assumption is too restrictive as they may be some intermediate states. Therefore, in this paper, we introduce four consistency states for an upper bound constraint. Specifically, in Section 2, we detail related work and problem analysis for multiple consistency states. In Section 3, we represent grid workflow time attributes. In
Section 4, we detail the four states. In Sections 5 and 6, we discuss corresponding verification and adjustment methods. In Section 7, we quantitatively evaluate our four-state approach by comparing it with the conventional two-state approach. The evaluation shows that our four-state approach can achieve better cost-effectiveness than the conventional two-state approach. In Section 8, we conclude our contributions and point out future work.

2. Related work and problem analysis

According to the literature, in [13], the authors use the modified Critical Path Method (CPM) to calculate temporal constraints and the work is one of the very few projects that consider temporal constraint reasoning at both build-time and run-time. In [21], the authors present a method for dynamic verification of absolute deadline constraints and relative deadline constraints. In [25], the authors propose a timed workflow model with considering the flow time, the time difference in a distributed execution environment. In [5, 6, 7], the authors investigate temporal dependency between temporal constraints and discuss its impact on temporal verification effectiveness and efficiency.

However, the above related work and some others such as [11] assume that an upper bound constraint only has two states: consistency and inconsistency. To distinguish with related concepts to be presented later by us, we rename them as CC (Conventional Consistency) and CI (Conventional Inconsistency) respectively. In grid workflow systems, this assumption is too restrictive. On one hand, a grid workflow is normally very complex and encompasses multiple administrative domains (organisations) over a wide area network. This results in difficult prediction on overall system performance, such as network latency [3]. Moreover, at the run-time instantiation stage, some extra activities, such as temporary data transfer activities, may be added to a grid workflow [12]. Therefore, at build-time, for some upper bound constraints, it is difficult to give accurate consistency setting. On the other hand, at the run-time execution stage, grid workflow execution environments are very dynamic. A grid service is not dedicated to one activity execution while sometimes more grid services may be able to participate in one activity execution at the same time. In addition, a transient grid service may have a changeable lifecycle. Hence, there could be some time redundancy saved potentially by the succeeding activities. With this time redundancy, some CI upper bound constraints may still be able to be recovered to CC. For those that are difficult to be recovered to CC, we can trigger the exception handling to handle them [16]. However, the exception handlings could vary depending on specific situations of CI. Some are simpler and more cost-effective while others are more complicated and costly. Since normally each exception handling causes some cost, we should separate them so that the exception handling triggered is appropriate, hence more cost-effective. Therefore, intermediate uncertain states between CC and CI need to be investigated.

Our initial work in [9] has discussed the intermediate uncertain states for fixed-time constraints. However, a fixed-time constraint is a special case of an upper bound constraint. Therefore, in this paper, we further extend the discussion to upper bound constraints. Correspondingly, we introduce four states for an upper bound constraint: SC (Strong Consistency), WC (Weak Consistency), WI (Weak Inconsistency), and SI (Strong Inconsistency). SC corresponds to CC and we divide CI into WC, WI and SI. We then develop their verification methods. We also discuss how to adjust WC to SC without triggering any exception handling as in the conventional work, and analyse why WI can rely on less costly exception handling.

3. Timed grid workflow representation

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 [11, 13]. We borrow some concepts from [5, 13, 21] such as maximum or minimum duration. We denote the maximum duration of an activity as $a_i$ and 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 $Rcd(a_i)$ respectively. $D(a_i)$, $d(a_i)$ can be obtained based on the past grid workflow execution history collected by grid workflow systems. $Rcd(a_i)$ includes delay time such as queuing delay, synchronisation delay or network latency. Normally, we have $d(a_i) \leq Rcd(a_i) \leq D(a_i)$.

In addition, we introduce mean duration to each activity. We denote the mean duration of activity $a_i$ as $M(a_i)$. The activity mean duration means that statistically the activity can be completed around its mean duration. According to [20, 23], we can apply some stochastic models such as Poisson distribution, exponential distribution and so on to obtain the mean duration for each activity. Normally, we have $d(a_i) \leq M(a_i) \leq D(a_i)$.

If there is a path from $a_i$ to $a_j$ ($i \neq j$), 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 $Rcd(a_i, a_j)$ respectively [13, 21]. And we denote the set of all activities from $a_i$ to $a_j$ as $[a_i, a_j]$. If there is an upper bound constraint between $a_i$ and $a_j$, we denote it as $UBC(a_i, a_j)$, its value as $ubv(a_i, a_j)$. For convenience, we only consider one execution path in the grid workflow without losing generality. As to a selective or parallel structure, for each branch, it is an execution path. For an iterative structure, from the start time to the end time, it is still an execution path. Therefore, for the

\[ \text{UBC}(a_i, a_j), ubv(a_i, a_j), D(a_i, a_j), d(a_i, a_j), M(a_i, a_j), Rcd(a_i, a_j) \]
selective/parallel/iterative structures, we can also apply the results achieved from one execution path. Correspondingly, between $a_i$ and $a_j$, $D(a_i, a_j)$ is equal to the sum of all activity maximum durations, and $d(a_i, a_j)$ is equal to the sum of all activity minimum durations, and $M(a_i, a_j)$ is equal to the sum of all activity mean durations.

There is a difference between $M(a_i)$ and $D(a_i)$. We define it as mean activity time redundancy.

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

4. SC, WC, WI and SI

We now define and explain SC, WC, WI and SI. For the comparison purpose, we also summarise definitions of CC and CI.

**Definition 2.** At build-time stage, $UBC(a_i, a_j)$ is said to be of SC if $D(a_i, a_j) \leq ubv(a_i, a_j)$, WC if $M(a_i, a_j) \leq ubv(a_i, a_j)$, and SI if $ubv(a_i, a_j) < d(a_i, a_j)$.

**Definition 3.** At run-time execution stage, at checkpoint $a_i$, between $a_i$ and $a_j$, $UBC(a_i, a_j)$ is said to be of SC if $ubv(a_i, a_j) + D(a_{p+1}, a_j) \leq ubv(a_i, a_j)$, WC if $ubv(a_i, a_j) + M(a_{p+1}, a_j) \leq ubv(a_i, a_j) < M(a_{p+1}, a_j) + D(a_{p+1}, a_j)$, and SI if $ubv(a_i, a_j) < ubv(a_i, a_j) + D(a_{p+1}, a_j)$.

**Definition 4.** At build-time stage, $UBC(a_i, a_j)$ is said to be of CC if $D(a_i, a_j) \leq ubv(a_i, a_j)$, and CI if $ubv(a_i, a_j) < D(a_i, a_j)$.

**Definition 5.** At run-time execution stage, at checkpoint $a_i$, between $a_i$ and $a_j$, $UBC(a_i, a_j)$ is said to be of CC if $ubv(a_i, a_j) + D(a_{p+1}, a_j) \leq ubv(a_i, a_j)$, and CI if $ubv(a_i, a_j) < ubv(a_i, a_j) + D(a_{p+1}, a_j)$.

Because at run-time instantiation stage, there are no specific activity execution times, the corresponding definitions of the stage are the same as those of the build-time stage, namely Definitions 2 and 4, hence omitted.

In addition, Definitions 3 and 5 do not consider those upper bound constraints which do not cover $a_i$. This is because grid workflow execution at $a_i$ does not affect the consistency of such upper bound constraints.

For clarity, we further compare SC, WC, WI and SI with CC and CI in Figure 1.

We now further explain SC, WC, WI and SI. We take the run-time instantiation stage, namely Definition 3, as the example. The explanation of the build-time stage definition, namely Definition 2, is similar.

- At checkpoint $a_i$, if $ubv(a_i, a_j) + D(a_{p+1}, a_j) \leq ubv(a_i, a_j)$, it means that $UBC(a_i, a_j)$ can be kept if succeeding activities can be completed by their respective maximum durations. Since activity maximum duration is carefully set and mostly should be kept, we define this state as SC.

In Table 1, we compare handling approaches for SC, WC, WI and SI by our research and by the conventional work.

![Figure 1. Definitions of SC, WC, WI and SI vs definitions of CC and CI at build-time and run-time execution stages](image-url)

- If $Rcd(a_i, a_j) + M(a_{p+1}, a_j) \leq ubv(a_i, a_j) < Rcd(a_i, a_j) + D(a_{p+1}, a_j)$, it means that, if succeeding activities take their respective maximum durations to complete, $UBC(a_i, a_j)$ will be violated. But if they just take about mean durations or less to complete, $UBC(a_i, a_j)$ can still be kept. Since statistically, an activity takes about its mean duration to complete, we define this state as WC. It means that with the control of succeeding activity execution based on activity mean duration, statistically $UBC(a_i, a_j)$ can still be kept.

- If $Rcd(a_i, a_j) + d(a_{p+1}, a_j) \leq ubv(a_i, a_j) < Rcd(a_i, a_j) + M(a_{p+1}, a_j)$, it means that, if succeeding activities take their respective mean durations or more to complete, $UBC(a_i, a_j)$ will be violated. However, if succeeding activities take about their respective minimum durations to complete, $UBC(a_i, a_j)$ may still be kept. According to the acquisition of the mean and minimum durations, this means that statistically, for most cases, $UBC(a_i, a_j)$ is difficult to be kept, and only for fewer cases where all succeeding activities can be completed by their respective minimum durations, $UBC(a_i, a_j)$ can be kept. Hence, we define this state as WI.

- If $ubv(a_i, a_j) < Rcd(a_i, a_j) + d(a_{p+1}, a_j)$, it means that, even if all succeeding activities can be completed by their respective minimum durations, $UBC(a_i, a_j)$ still cannot be kept. Accordingly to the setting of minimum durations, this means that mostly $UBC(a_i, a_j)$ cannot be kept. Therefore, we define this state as SI.
Table 1. Different handling approaches for SC, WC, WI and SI by our research and by conventional work

<table>
<thead>
<tr>
<th>Consistency states</th>
<th>SC</th>
<th>WC</th>
<th>WI</th>
<th>SI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our research</td>
<td>Doing nothing</td>
<td>Adjusting to SC without triggering exception handling</td>
<td>Triggering simpler exception handling than SI</td>
<td>Triggering more complex exception handling as conventional work</td>
</tr>
<tr>
<td>Conventional work</td>
<td>Treating as CC and doing nothing</td>
<td>Treating WC, WI and SI all as CI, and triggering the same exception handling</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

As shown in Table 1, for SC, like the conventional work, we need not do anything as mostly the corresponding upper bound constraints can be kept.

For WC, according to the definition explanation, by using mean activity time redundancy, statistically the corresponding upper bound constraints can still be kept. Therefore, we need not trigger exception handling as in the conventional work. Correspondingly, we will investigate how to adjust WC in Sections 5 and 6.

For WI, because for most cases it cannot be kept, we should call exception handling to adjust it to SC or WC.

For SI, normally, mostly it cannot be kept. So, we also resort to exception handling to adjust it to SC or WC.

However, WI exception handling is different from SI exception. On one hand, according to Definitions 2 and 3, the time deficit for WI to SC is smaller than that for SI. On the other hand, compared to SI, there are more cases where WI could still be kept like SC. So, the exception handling for WI can be simpler. Simpler exception handling will save more handling cost. In addition, if the current load of grid workflow systems is light and activities can be completed by their respective minimum durations, WI can be kept like SC even without triggering any exception handling. In summary, the introduction of WI can improve the exception handling in terms of its simplicity and cost effectiveness. However, the conventional work treats WI and SI (and even WC) by the same exception handling.

The specific investigation on the exception handling for WI and SI can be referred to [16] and is beyond the scope of this paper as it is not controlled only by temporal verification and normally is related to some other overall aspects such as overall QoS (Quality of Service) [16].

5. Build-time verification and WC adjustment

At build-time, we verify upper bound constraints and investigate how to use mean activity time redundancy to adjust WC to SC.

For each upper bound constraint, say $UBC(a_i, a_j)$, we compute $M(a_i, a_j)$, $d(a_i, a_j)$ and $D(a_i, a_j)$ according to Section 3. Then, we compare them with $ubv(a_i, a_j)$. Based on the comparison results and Definition 2, we can judge its consistency. We leave WI and SI for their respective exception handling. We now discuss WC adjustment.

Considering a WC upper bound constraint, say $UBC(a_i, a_j)$, to adjust it to SC, we have to compensate a time deficit, namely $D(a_i, a_j) - ubv(a_i, a_j)$. We allocate this deficit to activities between $a_i$ and $a_j$ by using their mean time redundancy. Considering an activity between $a_i$ and $a_j$, say $a_k$, we denote the deficit quota to be allocated to $a_k$ at build-time as $bdef_{jk}(a_k)$. We denote the current deficit quota that $a_k$ is holding as $def(a_k)$. Then, we propose:

$$bdef_{jk}(a_k) = [D(a_i, a_j) - ubv(a_i, a_j)] \frac{D(a_k) - M(a_k)}{\sum_{i\leq k \leq j} [D(a_i) - M(a_i)]} \quad (i \leq k \leq j) \quad (1)$$

In formula (1), we allocate the time deficit to $a_k$ based on the proportion of its mean time redundancy $D(a_k) - M(a_k)$ out of the overall mean time redundancy of all activities between $a_i$ and $a_j$. We can see that an activity with bigger mean time redundancy takes more time deficit quota. This is because the activity with a bigger mean time redundancy has more time to compensate the time deficit.

To be able to apply formula (1), we still have to solve two problems. The first problem is that for any activity $a_k$ between $a_i$ and $a_j$, we must ensure $bdef_{jk}(a_k) \leq D(a_k) - M(a_k)$. Otherwise, the available time for $a_k$ to complete is less than $M(a_k)$. Therefore, statistically, for most cases, the allocation will probably lead to new WC or WI or SI. This means that the allocation should not be applied. The second problem is that we must solve how to conduct multiple allocations. This is because there may be multiple WCs that cover some activities in common. For such common activities, there will be multiple allocations. So, we have to investigate how to conduct them jointly.

To solve the first problem, we derive Theorem 1 below.

**Theorem 1.** At build-time stage, for one WC upper bound constraint, say $UBC(a_i, a_j)$, if we allocate the time deficit to the activities covered by it according to formula (1), then, $\forall \ a_k \in [a_i, a_j]$, we have: $bdef_{jk}(a_k) \leq D(a_k) - M(a_k)$.

**Proof:** According to formula (1), to prove $bdef_{jk}(a_k) \leq D(a_k) - M(a_k)$, we only need to prove $D(a_k) - ubv(a_i, a_j) \leq \sum_{i\leq k \leq j} [D(a_i) - M(a_i)]$. In fact, $\sum_{i\leq k \leq j} [D(a_i) - M(a_i)] = D(a_i, a_j) - M(a_i, a_j)$. Therefore, we only need to prove $D(a_k) - ubv(a_i, a_j) \leq D(a_i, a_j) - M(a_i, a_j)$, namely $M(a_i, a_j) \leq ubv(a_i, a_j)$. Because $UBC(a_i, a_j)$ is of WC, according to Definition...
2, we do have $M(a_i, a_j) \leq ubv(a_i, a_j)$. Thus, the theorem holds.

Our approach to the second problem is as follows. Suppose now we are ready to allocate the time deficit for WC $UBC(a_i, a_j)$. Before we allocate $bdef_j(a_j)$ to $a_k$, we compare it with $def(a_j)$. If $def(a_j)$ is less, we replace the value of $def(a_j)$ with $bdef_j(a_j)$. Otherwise, we do nothing. We denote this approach as BTDA (Build-time Time Deficit Allocation). To be able to apply BTDA, on one hand, we must prove that all allocations are sufficient for all WC adjustments. Theorem 2 below supports this point.

On the other hand, similar to Theorem 1, we can also be less than or equal to $D(ak)-M(ak)$. Theorem 3 below supports this point.

**Theorem 2.** At build-time stage, given multiple WC upper bound constraints, if we allocate their time deficits to their respective activities according to formula (1) and BTDA, then statistically all allocations are sufficient for all of them to switch to SC.

**Proof:** For any $a_i$ of any WC upper bound constraint, say $UBC(a_i, a_j)$, after all allocations finish, according to BTDA, we have $bdef_j(a_j) \leq def(a_j)$. This means that at $a_i$, by taking $def(a_j)$, $UBC(a_i, a_j)$ can get more time to switch to SC than its own deficit allocation. Since $UBC(a_i, a_j)$ can switch to SC even only based on its own deficit allocation, based on multiple allocations, $UBC(a_i, a_j)$ can be easier to switch to SC. Thus, the theorem holds.

**Theorem 3.** At build-time stage, given multiple WC upper bound constraints, for each of them, say $UBC(a_i, a_j)$, if we allocate the time deficit to activities covered by it according to formula (1) and BTDA, then $\forall a_k \in [a_i, a_j]$, we also have: $def(a_k) \leq D(a_k)-M(a_k)$.

**Proof:** According to BTDA, $def(a_k)$ is equal to the maximum deficit quota allocated to $a_k$. According to Theorem 1, any deficit quota allocated to $a_k$ is less than or equal to $D(a_k)-M(a_k)$. Therefore, the maximum one must also be less than or equal to $D(a_k)-M(a_k)$. Thus, the theorem holds.

In conclusion, at build-time, we verify SC, WC, WI and SI according to Definition 2 and apply BTDA to adjust WC to SC while leaving WI and SI for their respective exception handling. Algorithm 1 depicts the whole process.

**Algorithm 1.** Temporal verification and WC adjustment at build-time stage for upper bound constraints

6. **Run-time verification and WC adjustment**

At the run-time instantiation stage, some temporary activities may be added to grid workflows. The consistency of upper bound constraints may be changed. So, we have to re-verify them. However, we do not have any specific activity execution times. The time information we can use is the same as that of the build-time stage. Therefore, the corresponding verification and WC adjustment are the same as those of the build-time stage. Hence, we simply omit corresponding discussion.

As discussed in Section 1, along with the grid workflow execution, some checkpoints are selected for conducting upper bound constraint verification [6, 21]. At checkpoint $a_p$, for each upper bound constraint which covers $a_p$, say $UBC(a_i, a_j)$, we verify it according to Definition 3. Similar to build-time, we leave WI and SI for their respective exception handling. We now discuss WC adjustment. Suppose that $UBC(a_i, a_j)$ is of WC at checkpoint $a_p$, comparing with SC, there is a time deficit that is $Red(a_p, a_{p+1}) + D(a_{p+1}, a_j) - ubv(a_i, a_j)$. To adjust WC, we allocate this deficit to the succeeding activities between $a_{p+1}$ and $a_j$. We are not able to allocate the deficit to other activities because all activities before $a_p$ have already completed and those after $a_i$ have nothing to do with the consistency of $UBC(a_i, a_j)$. Considering an activity between $a_{p+1}$ and $a_j$, say $a_k$, depending on the previous WC adjustments, $def(a_k)$ may be a positive value or zero. We denote the deficit quota to be allocated to $a_k$ as $rdef_k(a_k)$. Similar to the build-time discussion, we propose:
Similar to formula (1), we allocate the time deficit to ak based on the proportion of its mean time redundancy D(ak)-M(ak) out of the overall mean time redundancy of all activities between ap+1 and aj. The activity with a bigger mean time redundancy will carry more deficit quota. Due to the similar reason for the use of formula (1), to be able to apply formula (2), we also have to solve two problems. The first problem is that for any ai between ap+1 and aj, we must ensure rdef(i)(aj) ≤ D(aj)-M(aj). The second problem is how to conduct multiple allocations for multiple WC upper bound constraints which cover some activities in common. Theorem 4 below solves the first problem.

Theorem 4. At run-time execution stage, for one WC upper bound constraint, say UBC(ai, aj), if we allocate its time deficit to succeeding activities after checkpoint ap and covered by UBC(ai, aj) according to formula (2), then, ∀ ak ∈ [ap+1, aj], we have: rdef(i)(ak) ≤ D(aj) - M(aj).

Proof: Omitted as it is similar to that for Theorem 1.

To solve the second problem, similar to BTDA, we propose RTDA (Run-time Time Deficit Allocation) that is: before we allocate rdef(i)(ak) to ak, we compare it with def(ak). If def(ak) is less, we set rdef(i)(ak) to def(ak). Otherwise, we do nothing. Theorem 5 below shows that all allocations are sufficient for all WC adjustments. And Theorem 6 below ensures def(ak) ≤ D(aj) - M(aj).

Theorem 5. At run-time execution stage, given multiple WC upper bound constraints, if we allocate their time deficits to the succeeding activities after checkpoint ap and covered by them according to formula (2) and RTDA, the final allocation results are enough for all WC upper bound constraints to switch to SC.

Proof: Omitted as it is similar to that for Theorem 2.

Theorem 6. At run-time execution stage, given multiple WC upper bound constraints, for each of them, say UBC(ai, aj), if we allocate the time deficit to the succeeding activities after checkpoint ap and covered by UBC(ai, aj) according to formula (2) and RTDA, then, ∀ ak ∈ [ap+1, aj], we have: def(ak) ≤ D(aj) - M(aj).

Proof: Omitted as it is similar to that for Theorem 3.

Based on RTDA and Theorems 4, 5 and 6, we can derive another algorithm for upper bound constraint verification and WC adjustment at the run-time execution stage. However, due to the page limit and its similarity to Algorithm 1, we simply omit it.

7. Comparison and quantitative evaluation

Conventional upper bound constraint verification work treats WC, WI and SI as CI by the same exception handling. However, in this paper, by separating them, we can handle them differently. For WC, by deploying BTDA and RTDA, WC can switch to SC without triggering any exception handling. For WI, we can resort to less costly exception handling. Since every exception handling normally causes some cost such as compensating some completed activities [15], our four-state approach is more cost effective than the conventional two-state approach.

Now, we further conduct the 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 one. For simplicity, we only consider main exception handling cost which is spent on activities [16]. In a grid workflow gw, we denote the exception handling cost of ak as Ck(gw), the number of WC as N(gw), the number of WI as M(gw). In addition, we use Q(gw) to represent the number of activities addressed by the exception handling conducted by the conventional work for the jth WC or WI, and P(gw) to represent the number of activities addressed by the exception handling conducted by our work for the jth WI. And we denote total exception handling cost of the conventional work minus that of our research as DIFFtotal. Then, according to the discussion in this paper, statistically we have:

\[
\text{DIFF}_{\text{total}}(g_w) = N(g_w)Q(g_w) + M(g_w)P(g_w) + C(g_w) - C(k,g_w)
\]

To obtain an overall analysis, we analyse DIFFtotal(gw) in a statistical way. Therefore, we replace the corresponding variables in (3) with their respective mean values which can be achieved based on the past execution history. We denote mean values of Ck(gw), N(gw), M(gw), DIFFtotal(gw) as C, N, M, DIFF_{\text{total}} respectively. According to [20], statistically Q(gw) and P(gw) can be a kind of stochastic distribution. We take mean distribution without losing generality. For other stochastic distributions, the final conclusions would be similar. Correspondingly, we have Q(gw)=X*A, P(gw)=Y*B, A, stands for the number of activities covered by the jth WC. B, stands for the number of activities covered by the jth WI. X and Y are mean weights that depend on mean system load, available grid services and mean time deficit (0<X≤1, 0<Y≤1). According to Section 4, Y<X. Then, we can derive:

\[
\text{DIFF}_{\text{total}} = \sum_{j=1}^{P} C X A_j + \sum_{\infty} C (X - Y) B_j
\]
we can derive:

\[ A_i = \left[ P + (M - 1)Q \right] + iQ \]
\[ B_i = P + (i - 1)Q. \]

If we apply them to (4), then we have:

\[ \text{DIFF}_{\text{total}} = CXN[P + (M - 1 + \frac{(X-Y)}{2})Q] + C(X - Y)M[P + Q(M-1)] \]

We now take a set of specific values to see how (5) performs. We suppose that \( P=8, Q=3, X=1/2, Y=1/3, \) and \( C \) is equal to 1 cost unit. We also suppose that \( N \) can change from 0 to 10 and \( M \) can change from 0 to 20. The selection of these specific values does not affect our analysis because what we want is the trend of how \( \text{DIFF}_{\text{total}} \) changes to \( M \) and \( N \). Considering \( M=0, 2, 4, 6, 8, 10, \) with \( N \) changing, we list corresponding \( \text{DIFF}_{\text{total}} \) in Figure 2.

According to Figure 2, we can see that if \( M=0 \) and \( N=0, \) \( \text{DIFF}_{\text{total}} = 0. \) In fact, if \( M=0 \) and \( N=0, \) there is no any WC or WI. Then, according to the discussion in Section 4, the corresponding handling by our research is the same as that by the conventional work. Hence, \( \text{DIFF}_{\text{total}} = 0. \) In addition, we can also see that given fixed \( M, \) with \( N \) increasing, \( \text{DIFF}_{\text{total}} \) is increasing, and given fixed \( N, \) with \( M \) increasing, \( \text{DIFF}_{\text{total}} \) is increasing too. This means that the more WC or WI, the more exception handling cost saved by our research. Therefore, in overall terms, statistically our research can achieve better cost effectiveness than conventional work.

8. Conclusions and future work

In this paper, based on the analysis of grid workflow complexity and run-time dynamic grid service availability in grid workflow execution environments, four consistency states have been proposed. They are SC (Strong Consistency), WC (Weak Consistency), WI (Weak Inconsistency), and SI (Strong Inconsistency). Correspondingly, their verification methods have been presented. And for WC, algorithms on how to use mean activity time redundancy to adjust it to SC have been developed. According to these algorithms, no exception handling is needed for WC while in the conventional work it is a must. For WI, the reason why it can be treated by simpler exception handling than that used by the conventional work has been analysed. Finally, the quantitative evaluation has been conducted, which has shown that our four-state approach can achieve better cost effectiveness than the conventional two-state one.

Since our four-state approach has already allowed for specific features of dynamic grid workflow execution environments, by applying corresponding concepts, methods and algorithms derived in this paper, we can make upper bound constraint verification more applicable to dynamic grid workflow environments.

With these contributions, we can further investigate the exception handling for WI and SI, and checkpoint selection based on our four-state approach.

Acknowledgements

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

References


