

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 process 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 situation 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 i^{th} activity of a grid workflow 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 ($i \leq j$), 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 $Rcd(a_i, a_j)$ 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

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_k is defined as the difference between its maximum duration and its mean duration, namely $D(a_k) - M(a_k)$.

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 $M(a_i, a_j) \leq ubv(a_i, a_j) < D(a_i, a_j)$, WI if $d(a_i, a_j) \leq ubv(a_i, a_j) < M(a_i, a_j)$, and SI if $ubv(a_i, a_j) < d(a_i, a_j)$.

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

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_p between a_i and a_j , $UBC(a_i, a_j)$ is said to be of CC if $Rcd(a_i, a_p) + D(a_{p+1}, a_j) \leq ubv(a_i, a_j)$, and CI if $ubv(a_i, a_j) < Rcd(a_i, a_p) + D(a_{p+1}, a_j)$.

Because at run-time instantiation stage, there are no any 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_p . This is because grid workflow execution at a_p 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_p , if $Rcd(a_i, a_p) + D(a_{p+1}, a_j) \leq ubv(a_i, a_j)$, 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.

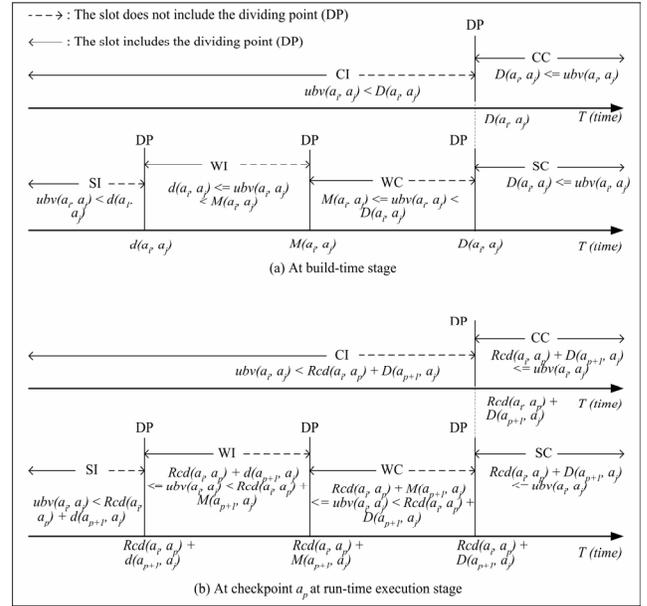


Figure 1. Definitions of SC, WC, WI and SI vs definitions of CC and CI at build-time and run-time execution stages

- If $Rcd(a_i, a_p) + M(a_{p+1}, a_j) \leq ubv(a_i, a_j) < Rcd(a_i, a_p) + D(a_{p+1}, a_j)$, 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 $Rcd(a_i, a_p) + d(a_{p+1}, a_j) \leq ubv(a_i, a_j) < Rcd(a_i, a_p) + M(a_{p+1}, a_j)$, 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 $ubv(a_i, a_j) < Rcd(a_i, a_p) + d(a_{p+1}, a_j)$, 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. According 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.

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

Table 1. Different handling approaches for SC, WC, WI and SI by our research and by conventional work

Consistency states	SC	WC	WI	SI
Our research	Doing nothing	Adjusting to SC without triggering exception handling	Triggering simpler exception handling than SI	Triggering more complex exception handling as conventional work
Conventional work	Treating as CC and doing nothing	Treating WC, WI and SI all as CI, and triggering the same exception handling for them all as that for SI		

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 $bdef_{ij}(a_k)$. We denote the current deficit quota that a_k is holding as $def(a_k)$. Then, we propose:

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

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 $bdef_{ij}(a_k) \leq D(a_k) - M(a_k)$. 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: $bdef_{ij}(a_k) \leq D(a_k) - M(a_k)$.

Proof: According to formula (1), to prove $bdef_{ij}(a_k) \leq D(a_k) - M(a_k)$, we only need to prove $D(a_i, a_j) - ubv(a_i, a_j) \leq \sum_{l=i}^j [D(a_l) - M(a_l)]$. In fact, $\sum_{l=i}^j [D(a_l) - M(a_l)] = D(a_i, a_j) - M(a_i, a_j)$. Therefore, we only need to prove $D(a_i, a_j) - ubv(a_i, a_j) \leq D(a_i, a_j) - M(a_i, a_j)$, 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 $bdef_{ij}(a_k)$ to a_k , we compare it with $def(a_k)$. If $def(a_k)$ is less, we replace the value of $def(a_k)$ with $bdef_{ij}(a_k)$. 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 $bdef_{ij}(a_k)$, we must ensure $def(a_k) \leq D(a_k) - M(a_k)$. 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_k of any WC upper bound constraint, say $UBC(a_i, a_j)$, after all allocations finish, according to BTDA, we have $bdef_{ij}(a_k) \leq def(a_k)$. This means that at a_k , by taking $def(a_k)$, $UBC(a_i, a_j)$ 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.

symbol Definitions:

ArrayUB: an array of all upper bound constraint indexes;

end Definitions

Input: ArrayUB, values of all upper bound constraints, activity maximum, minimum and mean durations;

Output: SC, WC, WI and SI report; WC adjustment to SC;

while (not end of ArrayUB) **do**

Select current upper bound constraint, say $UBC(a_i, a_j)$;

Compute $M(a_i, a_j)$ and $D(a_i, a_j)$;

if $(D(a_i, a_j) \leq ubv(a_i, a_j))$ **then**

Output ‘ $UBC(a_i, a_j)$ is of SC’

else if $(d(a_i, a_j) \leq ubv(a_i, a_j) < M(a_i, a_j))$ **then**

Output ‘ $UBC(a_i, a_j)$ is of WI’ ;

Call WI exception handling to change to SC or WC

```

else if  $(ubv(a_i, a_j) < d(a_i, a_j))$  then
    Output ‘ $UBC(a_i, a_j)$  is of SI’ ;
    Call SI exception handling to change to SC or WC
end if
// Both WI and SI exception handlings may adjust  $d(a_i, a_j)$ ,
//  $D(a_i, a_j)$  or  $ubv(a_i, a_j)$ , and are beyond the scope of this
// paper. However, ArrayUB remains unchanged.
if  $(M(a_i, a_j) \leq ubv(a_i, a_j) < D(a_i, a_j))$  then
    // may include WCs derived by WI and SI exception
    // handlings above
    Output ‘ $UBC(a_i, a_j)$  is of WC’ ;
    Copy all activities covered by  $UBC(a_i, a_j)$  to
        SetOfActivity;
    while (SetOfActivity not empty) do
        // WC adjustment according to BTDA
        Pop up an activity, say  $a_k$ ;
        Compute  $def_{ij}(a_k)$  according to formula (1);
        if  $(def(a_k) < def_{ij}(a_k))$  then
             $def(a_k) = def_{ij}(a_k)$ ;
        end if
    end while
end if
end while

```

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 $Rcd(a_i, a_p) + D(a_{p+1}, a_j) - ubv(a_i, a_j)$. 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_j 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, $def(a_k)$ ($p+1 \leq k \leq j$) may be a positive value or zero. We denote the deficit quota to be allocated to a_k as $rdef_{ij}(a_k)$. Similar to the build-time discussion, we propose:

$$rdef_{ij}(a_k) = [Rcd(a_i, a_p) + D(a_{p+1}, a_j) - ub(a_i, a_j)] \frac{D(a_k) - M(a_k)}{\sum_{l=p+1}^j [D(a_l) - M(a_l)]} \quad (p+1 \leq k \leq j) \quad (2)$$

Similar to 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_{p+1} and a_j . 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 a_k between a_{p+1} and a_j , we must ensure $rdef_{ij}(a_k) \leq D(a_k) - M(a_k)$. 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 $UBC(a_i, a_j)$, if we allocate its time deficit to succeeding activities after checkpoint a_p and covered by $UBC(a_i, a_j)$ according to formula (2), then, $\forall a_k \in [a_{p+1}, a_j]$, we have: $rdef_{ij}(a_k) \leq D(a_k) - M(a_k)$.

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 $rdef_{ij}(a_k)$ to a_k , we compare it with $def(a_k)$. If $def(a_k)$ is less, we set $rdef_{ij}(a_k)$ to $def(a_k)$. Otherwise, we do nothing. Theorem 5 below shows that all allocations are sufficient for all WC adjustments. And Theorem 6 below ensures $def(a_k) \leq D(a_k) - M(a_k)$.

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 a_p 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 $UBC(a_i, a_j)$, if we allocate the time deficit to the succeeding activities after checkpoint a_p and covered by $UBC(a_i, a_j)$ according to formula (2) and RTDA, then, $\forall a_k \in [a_{p+1}, a_j]$, we have: $def(a_k) \leq D(a_k) - M(a_k)$.

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 a_k as $C_k(gw)$, the number of WC as $N(gw)$, the number of WI as $M(gw)$. In addition, we use $Q_j(gw)$ to represent the number of activities addressed by the exception handling conducted by the conventional work for the j^{th} WC or WI, and $P_i(gw)$ to represent the number of activities addressed by the exception handling conducted by our work for the i^{th} WI. And we denote total exception handling cost of the conventional work minus that of our research as $DIFF_{total}(gw)$. Then, according to the discussion in this paper, statistically we have:

$$DIFF_{total}(gw) = \sum_{i=1}^{N(gw)} Q_i(gw) \sum_{k=1}^{Q_i(gw)} C_k(gw) + \sum_{j=1}^{M(gw)} Q_j(gw) \sum_{s=1}^{Q_j(gw)} C_s(gw) - \sum_{i=1}^{M(gw)} P_i(gw) \sum_{m=1}^{P_i(gw)} C_m(gw) \quad (3)$$

To obtain an overall analysis, we analyse $DIFF_{total}(gw)$ 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 $C_k(gw)$, $N(gw)$, $M(gw)$, $DIFF_{total}(gw)$ as C , N , M , $DIFF_{total}$ respectively. According to [20], statistically $Q_j(gw)$ and $P_i(gw)$ 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 $Q_j(gw) = X * A_j$, $P_i(gw) = Y * B_i$. A_j stands for the number of activities covered by the j^{th} WC. B_i stands for the number of activities covered by the i^{th} WI. X and Y are mean weights that depend on mean system load, available grid services and mean time deficit ($0 < X \leq 1$, $0 < Y \leq 1$). According to Section 4, $Y < X$. Then, we can derive:

$$DIFF_{total} = \sum_{j=1}^N C X A_j + \sum_{i=1}^M C (X - Y) B_i \quad (4)$$

We consider the most complicated case where upper bound constraints are nested one after another. We suppose that the first upper bound constraint covers P activities. We assume that the distance between any two adjacent upper bound constraints is the same, denoted as Q . Because (4) has nothing to do with SI, we assume that there is no SI. In addition, according to the discussion of

the temporal dependency in [5, 7], all nested upper bound constraints after a WC one must be of WC or even SC, and all upper bound constraints after a SC one must be of SC. The detailed explanation can be found in [5, 7]. Therefore, we can derive: $A_j = [P+(M-1)Q]+jQ$ and $B_i = P+(i-1)Q$. If we apply them to (4), then we have:

$$DIFF_{total} = CXN[P + (M - 1 + \frac{(N+1)}{2})Q] + C(X - Y)M[P + Q\frac{(M-1)}{2}] \quad (5)$$

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 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 $DIFF_{total}$ changes to M and N . Considering $M=0, 2, 4, 6, 8, 10$, with N changing, we list corresponding $DIFF_{total}$ in Figure 2.

According to Figure 2, we can see that if $M=0$ and $N=0$, $DIFF_{total} = 0$. 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, $DIFF_{total}$ is 0. In addition, we can also see that given fixed M , with N increasing, $DIFF_{total}$ is increasing, and given fixed N , with M increasing, $DIFF_{total}$ 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.

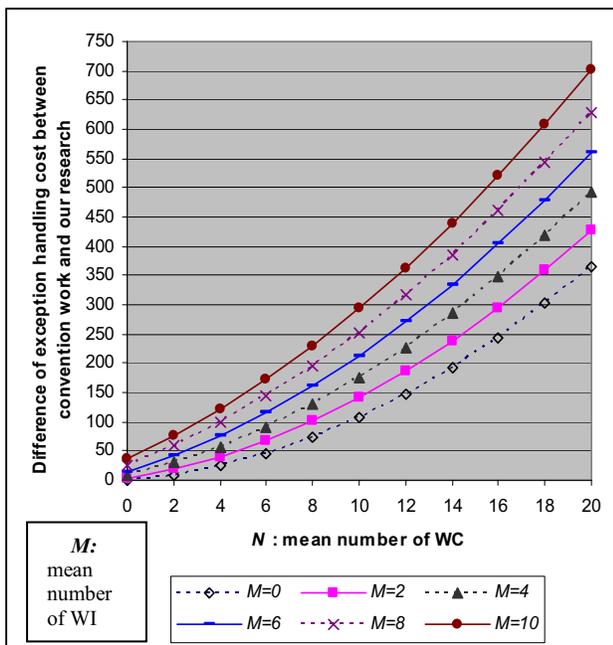


Figure 2. Change trend of exception handling cost difference between conventional work and our research

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

- [1] D. Abramson, J. Kommineni, J.L. McGregor, and J.Katzfey, "An Atmospheric Sciences Workflow and Its Implementation with Web Services", In Proc. of the 4th International Conference on Computational Science, Part I, Springer Verlag, Krakow, Poland, 2004, LNCS 3036, pp. 164-173.
- [2] K. Amin, G.V. Laszewski, M. Hategan, N.J. Zaluzec, S. Hampton, and A. Rossi, "GridAnt: A Client-controllable Grid Workflow", In Proc. of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04), Hawaii, Jan. 2004, pp. 210-219.
- [3] J. Cao, S.A. Jarvis, S. Saini, and G.R. Nudd, "GridFlow: Workflow Management for Grid Computing", In Proc. of the 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003), Tokyo, May 2003, pp. 198-205.

- [4] D. Cybok, "A Grid Workflow Infrastructure", In Proc. of Workflow in Grid Systems Workshop in GGF10, Berlin, Germany, Mar. 2004.
- [5] J. Chen, and Y. Yang, "Temporal Dependency for Dynamic Verification of Temporal Constraints in Workflow Systems", In Proc. of 3rd International Conference on Grid and Cooperative Computing, Springer-Verlag, 2004, LNCS 3251, pp. 1005-1008.
- [6] J. Chen, Y. Yang, and T.Y. Chen, "Dynamic Verification of Temporal Constraints on-the-fly for Workflow Systems", In Proc. of the 11th Asia-Pacific Software Engineering Conference (APSEC2004), IEEE CS Press, Busan, Korea, Nov./Dec. 2004, pp. 30-37.
- [7] J. Chen, and Y. Yang, "Temporal Dependency for Dynamic Verification of Fixed-date Constraints in Grid Workflow Systems", In Proc. of the 7th Asia Pacific Web Conference, Springer-Verlag, 2004, LNCS 3399, pp. 820-831.
- [8] J. Chen, and Y. Yang, "An Activity Completion Duration based Checkpoint Selection Strategy for Dynamic Verification of Fixed-time Constraints in Grid Workflow Systems", In Proc. of 2nd International Conference on Grid Service Engineering and Management, Sept. 2005, LNI P-69, pp. 296-310.
- [9] J. Chen, and Y. Yang, "Flexible Temporal Consistency for Fixed-time Constraint Verification in Grid Workflow Systems", In Proc. of the 4th International Conference on Grid and Cooperative Computing (GCC2005), Springer-Verlag, Nov./Dec. 2005, LNCS, to appear, <http://www.it.swin.edu.au/personal/yyang/papers/GCC2005.pdf>.
- [10] J. Chen, and Y. Yang, "A Minimum Proportional Time Redundancy based Checkpoint Selection Strategy for Dynamic Verification of Fixed-time Constraints in Grid Workflow Systems", In Proc. of the 12th Asia-Pacific Software Engineering Conference (APSEC2005), IEEE CS Press, Taipei, Taiwan, Dec. 2005, to appear, <http://www.it.swin.edu.au/personal/yyang/papers/APSEC2005.pdf>.
- [11] S. Chinn, and G. Madey, "Temporal Representation and Reasoning for Workflow in Engineering Design Change Review", IEEE Transactions on Engineering Management, 2000, 47(4), pp. 485-492.
- [12] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, and K. Vahi, "Mapping Abstract Complex Workflows onto Grid Environments", Journal of Grid Computing, 2003, 1(1), pp. 9-23.
- [13] J. Eder, E. Panagos, and M. Rabinovich, "Time Constraints in Workflow Systems", In Proc. of the 11th International Conference on Advanced Information Systems Engineering (CAiSE'99), 1999, LNCS 1626, pp. 286-300.
- [14] T. Fahringer, S. Pillana, and A. Villazon, "A-GWL: Abstract Grid Workflow Language", In Proc. of the 4th International Conference on Computational Science, Part III, Springer Verlag, Krakow, Poland, 2004, LNCS 3038, pp. 42-49.
- [15] I. Foster, C. Kesselman, J. Nick, and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", In Proc. of the 5th Global Grid Forum Workshop (GGF5), Edinburgh, Scotland, July 2002.
- [16] C. Hagen, and G. Alonso, "Exception Handling in Workflow Management Systems", IEEE Transactions on Software Engineering, 2000, 26(10), pp. 943-958.
- [17] Y. Huang, "JISGA: A JINI-BASED Service-Oriented Grid Architecture", The International Journal of High Performance Computing Applications, 2003, 17(3), pp. 317-327.
- [18] S. Krishnan, P. Wagstrom, and G.V. Laszewski, "GSFL: A Workflow Framework for Grid Services", Technical Report, Argonne National Laboratory, Argonne, U.S.A., 2002.
- [19] H. Li, Y. Yang, and T.Y. Chen, "Resource Constraints Analysis of Workflow Specifications", The Journal of Systems and Software, 2004, 73(2), pp. 271-285.
- [20] Z. Liu, "Performance Analysis of Stochastic Timed Petri Nets Using Linear Programming Approach", IEEE Transactions on Software Engineering, 1998, 11(24), pp. 1014-1030.
- [21] O. Marjanovic, and M.E. Orlowska, "On Modeling and Verification of Temporal Constraints in Production Workflows", Knowledge and Information Systems, 1999, 1(2), pp. 157-192.
- [22] D.R. Simpson, N. Kelly, P.V. Jithesh, P. Donachy, T.J. Harmer, R.H. Perrott, J. Johnston, P. Kerr, M. McCurley, and S. McKee, "GeneGrid: A Practical Workflow Implementation for a Grid Based Virtual Bioinformatics Laboratory", In Proc. of the UK e-Science All Hands Meeting 2004 (AHM04), 2004, pp. 547-554.
- [23] J.H. Son, and M.H. Kim, "Improving the Performance of Time-constrained Workflow Processing", The Journal of Systems and Software, 2001, 58(3), pp. 211-219.
- [24] J. Yu, and R. Buyya, "A Novel Architecture for Realizing Grid Workflow Using Tuple Spaces", In Proc. of the 5th IEEE/ACM International Workshop on Grid Computing (GRID'04) - Volume 00, IEEE CS Press, Los Alamitos, USA, Nov. 2004, pp. 119-128.
- [25] H. Zhuge, T. Cheung, and H. Pung, "A Timed Workflow Process Model", The Journal of Systems and Software, 2001, 55(3), pp. 231-243.