Assigning Local Fixed-time Constraints in Grid Workflow Systems

Jinjun Chen, Yun Yang
CITR – Centre for Information Technology Research
Faculty of Information and Communication Technologies
Swinburne University of Technology
PO Box 218, Hawthorn, Melbourne, Australia 3122
{jchen, yyang}@ict.swin.edu.au

Abstract

Many complex grid workflow processes often have only one global fixed-time constraint at the last activity. Since grid workflow execution normally lasts a long time, local control is very important to ensure the overall temporal correctness. With only one global fixed-time constraint, we cannot control grid workflow execution locally in terms of time. As a result, for any temporal violations, we must handle them globally with the consideration of all activities. This would impact the overall cost-effectiveness of grid workflow execution. Therefore, in this paper, we investigate how to assign local fixed-time constraints within the global one so that we can control grid workflow execution locally. Corresponding assigning and verification methods are developed. With local fixed-time constraints, we can achieve better cost-effectiveness.

1. Introduction

In the grid architecture, a grid workflow system is facilitated to support large-scale sophisticated scientific or business processes in a variety of complex e-science or e-business applications such as climate modelling, disaster recovery, medical surgery, high energy physics, or international stock market modelling [1, 17, 22]. Such sophisticated processes are modelled or redesigned as grid workflow specifications at build-time stage which normally contain a large number of computation or data intensive activities [6, 16, 18], then, instantiated at run-time instantiation stage [7, 14, 20], and finally, are executed at run-time execution stage by facilitating the super computing and data sharing ability of underlying grid infrastructure [7, 19, 20].

In reality, complex scientific or business processes are normally time constrained [2, 4, 5]. Consequently, when they are modelled or redesigned as grid workflow specifications at build-time stage, fixed-time constraints are often set as well [1, 5, 9]. A fixed-time constraint at an activity is an absolute time value by which the activity must be completed [5, 9]. For example, a climate modelling grid workflow must be completed by a scheduled time [1], say 6:00pm, so that the weather forecast can be broadcast on time at a later time. Here, 6:00pm is a fixed-time constraint. Temporal verification is conducted to check if all fixed-time constraints are consistent. Temporal verification is carried out at build-time, run-time instantiation and run-time execution stages [15, 22]. Especially at the run-time execution stage, some checkpoints are often selected so that we only need to conduct temporal verification at such checkpoints rather than at all activity points [9, 10, 13, 25]. The detailed discussion about checkpoint selection is outside the scope of this paper and can be found in some other references such as [9, 10, 13, 25]. Here, we simply assume that all checkpoints have already been selected.

Many grid workflow processes such as climate modelling often have only one global fixed-time constraint [1]. A grid workflow execution normally lasts a long time as it often contains hundreds of thousands of activities [1, 3, 24]. Correspondingly, local control is very important to ensure the overall temporal correctness. With only one global fixed-time constraint, the local time control cannot be controlled easily. As a result, we may find the temporal violation at the latest execution stage and it might be too late to take any preventive actions. Then, the execution results may be useless and the overall cost-effectiveness would be impacted. Therefore, we must investigate how to assign local fixed-time constraints within the global one so that we can control grid workflow execution locally. Some related work has been done on the reasoning about fixed-time constraints. Al-Ali et al. [2] analyse QoS (Quality of Service) including temporal QoS in distributed grid applications including grid workflow applications. Buyya et al. [5] discuss grid economy issues including the temporal aspect in grid architecture. Chen et al. [9, 10, 13] address the checkpoint selection issue for conducting temporal verification. They also analyse temporal dependency between fixed-time constraints in grid workflow systems in [8]. Eder et al. [15] use a modified Critical Path Method (CPM) to conduct temporal reasoning. Marjanovic et al. [22] introduce minimum and maximum durations to each activity in workflow specifications. Based on this, they present a method for dynamic temporal verification.

However, the above related work does not pay sufficient attention to the assigning of local fixed-time constraints in order to control grid workflow execution locally. Therefore, in this paper, we systematically investigate how to assign local fixed-time constraints within the global one. We also develop corresponding assigning and verification methods.

There is another type of temporal constraints named upper bound constraints. An upper bound constraint between two activities is a relative time value so that the duration between them is less than or equal to it [15]. However, assigning local upper bound constraints is different from assigning local fixed-time constraints. This is because an upper bound constraint is relative while a fixed-time constraint is absolute. All fixed-time constraints have the same start point, i.e. the start activity of the grid workflow, and are nested one after another. But different upper bound constraints could have different start points and may not be nested one after another. In this paper, we focus on how to assign local fixed-time constraints. The corresponding discussion for upper bound constraints would be conducted somewhere else.

The remainder of the paper is organised as follows. In Section 2, we represent some time attributes of grid workflows. In Section 3, we discuss how to assign local fixed-time constraints at build-time and run-time instantiation stages. In Section 4, we investigate how to verify and adjust local fixed-time constraints at run-time execution stage. In Section 5, we conduct a simple comparison to intuitively demonstrate that we can achieve better cost-effectiveness with local fixed-time constraints than only with the global fixed-time constraint. Finally in Section 6, we conclude our contributions and point out future work.

2. Timed grid workflow representation

According to [21, 23], based on the directed network graph (DNG) concept, a grid workflow can be represented as a DNG-based grid workflow graph, where nodes correspond to activities and edges correspond to dependencies between activities. In [21, 23], the iterative structure is nested in an activity that has an exit condition defined for iterative purposes. Accordingly, the corresponding DNG-based grid workflow graph is structurally acyclic[1].

To represent activity time attributes, we borrow some concepts from [22, 25] such as maximum or minimum duration as a basis. We denote the i^{th} activity of a grid workflow gw as a_i. We denote the expected time from which the specification of grid workflow gw will come into effect as $Cie(gw)$. From $Cie(gw)$, the specification of grid workflow gw can be used. For each a_i, we denote its maximum duration, mean duration, minimum duration, run-time start time, run-time end time and run-time completion duration as $D(a_i)$, $M(a_i)$, $d(a_i)$, $S(a_i)$, $E(a_i)$ and $R(a_i)$ respectively. $M(a_i)$ means that statistically a_i can be completed around its mean duration. Other time attributes are self-explanatory. According to [22, 25], $D(a_i)$, $M(a_i)$, $d(a_i)$ can be obtained based on the past execution history. The past execution history covers the delay time incurred at a_i such as the setup delay, queuing delay, synchronisation delay, network latency and so on. The detailed discussion of $D(a_i)$, $M(a_i)$, $d(a_i)$ is outside the scope of this paper and can be referred to [22, 25]. For a specific execution of a_i, the delay time is included in $R(a_i)$. Normally, we have $d(a_i) \leq M(a_i) \leq D(a_i)$ and $d(a_i) \leq R(a_i) \leq D(a_i)$. If there is a fixed-time constraint set at a_i, we denote it as $FTC(a_i)$ and its value as $ftv(a_i)$. If there is a path from a_i to a_j ($i<j$), we denote the maximum duration, mean duration, minimum duration, run-time completion duration between them as $D(a_i, a_j)$, $M(a_i, a_j)$, $d(a_i, a_j)$ and $R(a_i, a_j)$ respectively. Normally we have $d(a_i, a_j) \leq M(a_i, a_j) \leq D(a_i, a_j)$ and $d(a_i, a_j) \leq R(a_i, a_j) \leq D(a_i, a_j)$.

For convenience of the discussion, we only consider one execution path in the acyclic DNG-based grid workflow graph without losing generality. As to a selective or parallel structure, each branch is an execution path. Therefore, we can equally apply the results achieved in this paper to each branch directly. In overall terms, for a grid workflow containing many parallel, selective and/or mixed structures, firstly, we treat each structure as an activity. Then, the whole grid workflow will be an overall execution path and we can apply the results achieved in this paper to it. Secondly, for every structure, for each of its branches, we continue to apply the results achieved in this paper. Thirdly, we carry out this recursive process until we complete all branches of all structures. Correspondingly, between a_i and a_j, $D(a_i, a_j)$ is equal to the sum of all activity maximum durations, $M(a_i, a_j)$ is equal to the sum of all activity mean durations, $d(a_i, a_j)$ is equal to the sum of all activity minimum durations.

Besides the above time attributes, four temporal consistency states have been identified and defined in [11, 12]. They are SC (Strong Consistency), WC (Weak Consistency), WI (Weak Inconsistency) and SI (Strong Inconsistency). We summarise their definitions in Definitions 1, 2 and 3. The detailed discussion about the four consistency states can be found in [11, 12].

Definition 1. At 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 $ftv(a_i) - Cie(gw) < d(a_1, a_i)$.

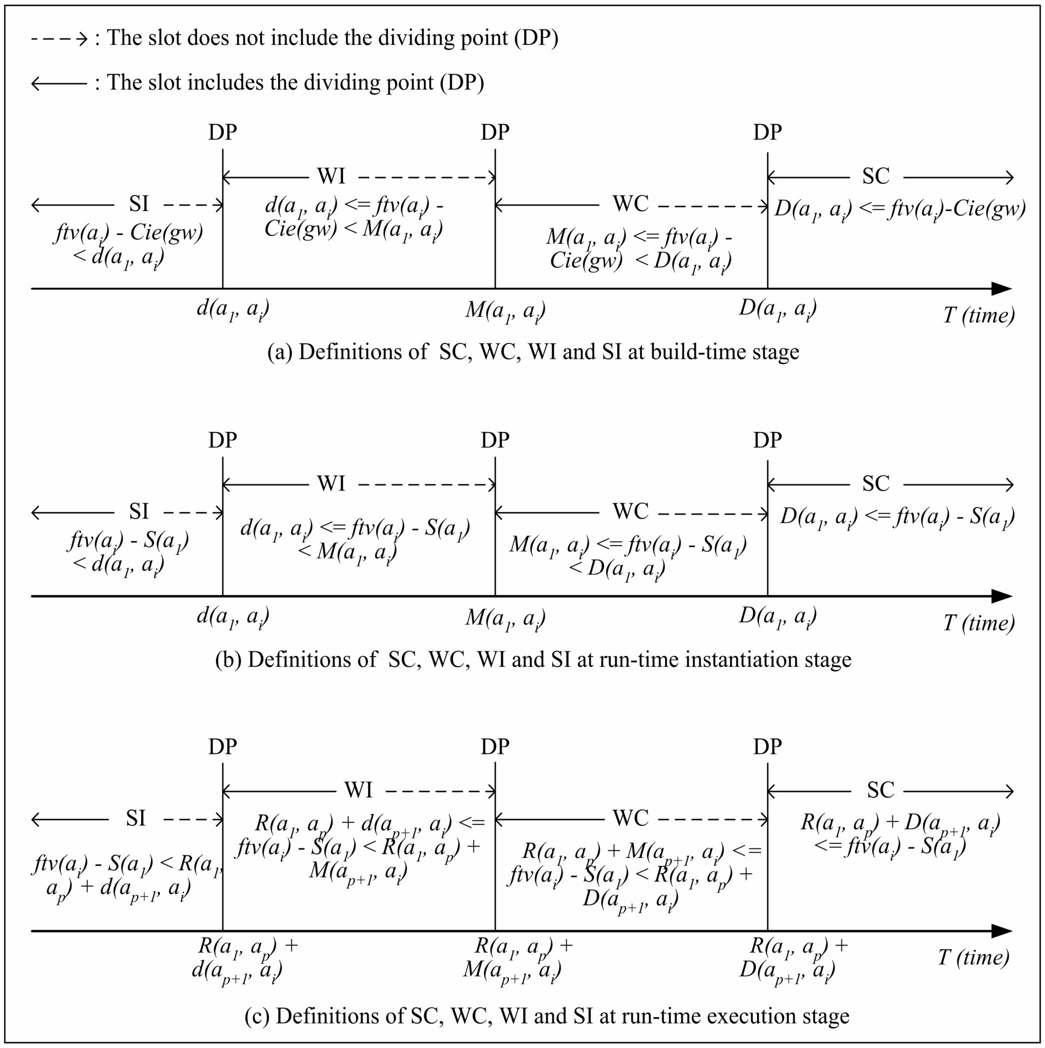
Definition 2. At build-time stage, $FTC(a_i)$ is said to be of SC if $D(a_1, a_i) \leq ftv(a_i) - S(a_1)$, WC if $M(a_1, a_i) \leq ftv(a_i) -$

[1] Refer to [23, 25] for more detail.

$S(a_1) < D(a_1, a_i)$, WI if *$d(a_1, a_i) \le ftv(a_i) - S(a_1) < M(a_1, a_i)$*, and SI if *$ftv(a_i) - S(a_1) < d(a_1, a_i)$*.

Definition 3. At run-time execution stage, at checkpoint a_p which is either before or at a_i ($p \le i$), $FTC(a_i)$ is said to be of SC if $R(a_1, a_p) + D(a_{p+1}, a_i) \le ftv(a_i) - S(a_1)$, WC if $R(a_1, a_p) + M(a_{p+1}, a_i) \le ftv(a_i) - S(a_1) < R(a_1, a_p) + D(a_{p+1}, a_i)$, WI if $R(a_1, a_p) + d(a_{p+1}, a_i) \le ftv(a_i) - S(a_1) < R(a_1, a_p) + M(a_{p+1}, a_i)$, and SI if $ftv(a_i) - S(a_1) < R(a_1, a_p) + d(a_{p+1}, a_i)$.

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



- - -> : The slot does not include the dividing point (DP)
<—— : The slot includes the dividing point (DP)
DP
DP
DP
WI
SC
SI
$d(a_1, a_i) <= ftv(a_i) - Cie(gw) < M(a_1, a_i)$
WC
$D(a_1, a_i) <= ftv(a_i)-Cie(gw)$
$ftv(a_i) - Cie(gw) < d(a_1, a_i)$
$M(a_1, a_i) <= ftv(a_i) - Cie(gw) < D(a_1, a_i)$
$d(a_1, a_i)$
$M(a_1, a_i)$
$D(a_1, a_i)$
T (time)
(a) Definitions of SC, WC, WI and SI at build-time stage
$d(a_1, a_i) <= ftv(a_i) - S(a_1) < M(a_1, a_i)$
$D(a_1, a_i) <= ftv(a_i) - S(a_1)$
$ftv(a_i) - S(a_1) < d(a_1, a_i)$
$M(a_1, a_i) <= ftv(a_i) - S(a_1) < D(a_1, a_i)$
(b) Definitions of SC, WC, WI and SI at run-time instantiation stage
$R(a_1, a_p) + d(a_{p+1}, a_i) <= ftv(a_i) - S(a_1) < R(a_1, a_p) + M(a_{p+1}, a_i)$
$R(a_1, a_p) + D(a_{p+1}, a_i) <= ftv(a_i) - S(a_1)$
$ftv(a_i) - S(a_1) < R(a_1, a_p) + d(a_{p+1}, a_i)$
$R(a_1, a_p) + M(a_{p+1}, a_i) <= ftv(a_i) - S(a_1) < R(a_1, a_p) + D(a_{p+1}, a_i)$
$R(a_1, a_p) + d(a_{p+1}, a_i)$
$R(a_1, a_p) + M(a_{p+1}, a_i)$
$R(a_1, a_p) + D(a_{p+1}, a_i)$
(c) Definitions of SC, WC, WI and SI at run-time execution stage

Figure 1. Definitions of SC, WC, WI and SI for fixed-time constraints

In this paper, we focus on SC only. The corresponding discussion for WC, WI and SI is similar as according to Definitions 1, 2 and 3, their definition styles are similar to that of SC.

3. Assigning local fixed-time constraints at build-time and run-time instantiation stages

We denote the global fixed-time constraint at the last activity as *F* and its value as *ftv(F)*. At build-time stage, we first should ensure *F* is of SC by verifying and adjusting it according to Definition 1. Then, we consider how to assign local fixed-time constraint within *F*. Based on the past execution history, we can summarise at which activities temporal violations often happen. Then, at such activities, we must take special control in order to ensure overall temporal correctness. Accordingly, at each of such activities we should assign a fixed-time. We suppose there are *N* such activities and we denote them as a_{j_1}, a_{j_2}, ... , a_{j_N} respectively ($j_1<j_2<...<j_N$). Correspondingly, we need to assign *N* local fixed-time constraints at a_{j_1}, a_{j_2}, ... , a_{j_N} respectively. We denote them as F_1, F_2, ... , F_N, and their

values as *ftv(F₁), ftv(F₂), ... , and ftv(F_N)*, i.e. F_i at a_{j_i} (*i=1, 2, ..., N*). When grid workflow *gw* is executed, it starts from a_1 and then reaches each fixed-time constraint gradually during the execution. Correspondingly, we can view a_1 as the start point of each fixed-time constraint. That is to say, F_1, F_2, ... , and F_N have the same start point [5, 9]. We suppose totally there are *T* activities covered by *F* ($N \le T$). Then, if *F* is of SC, we have a time redundancy: $[ftv(F) - Cie(gw)] - D(a_1, a_T)$. This time redundancy can be used to tolerate certain time deviation which might be incurred by the grid workflow execution. By allocating this time redundancy to activities covered by *F* so that each activity can hold a time quota, we can derive F_1, F_2, ... , F_N as follows.

Among *T* activities covered by *F*, we first sort all $D(a_s) - M(a_s)$ ($s = 1, 2, 3, ... , T$) in ascending order. Correspondingly, we will get a sorting list which contains all $D(a_s) - M(a_s)$ ($s = 1, 2, 3, ... , T$). We denote the list as *L* and the items in *L* as L_1, L_2, ... , L_T. If $D(a_s) - M(a_s)$ is ranked No. *k* in *L*, namely L_k, then we propose formula (1) to allocate $[ftv(F) - Cie(gw)] - D(a_1, a_T)$ to each of the *T* activities. We denote the time quota allocated to a_s as $TQ(a_s)$.

$$TQ(a_s) = [ftv(F) - Cie(gw) - D(a_1, a_T)] \frac{L_{T-k+1}}{\sum_{l=1}^{T}[D(a_l) - M(a_l)]} \qquad (1 \le k \le T) \qquad (1)$$

In (1), we allocate $[ftv(F) - Cie(gw)] - D(a_1, a_T)$ to activities covered by *F* based on the difference between activity maximum duration and activity mean duration. The activity with a bigger difference will be allocated a smaller quota. This is because statistically, an activity can be completed around its mean duration. Therefore, the activity with a bigger difference between its maximum duration and its mean duration has more time to compensate the possible time deviation incurred by the abnormal grid workflow execution. Hence, we should allocate a smaller quota to it.

After we allocate $[ftv(F) - Cie(gw)] - D(a_1, a_T)$ to activities covered by *F*, we can derive the values of F_1, F_2, ... , and F_N at a_{j_1}, a_{j_2}, ... , a_{j_N} respectively. Suppose we now considering F_i at a_{j_i}, we derive its value by formula (2).

$$ftv(F_i) = \sum_{s=1}^{j_i}[TQ(a_s) + D(a_s)] \qquad (i=1, 2, 3, ... , N) \qquad (2)$$

The basic relationships between *F* and F_1, F_2, ... , F_N are shown in Figure 2. We can see that they are nested one after another. This is because all fixed-time constraints including the global one has the same start point, i.e. the start activity of the whole grid workflow.

At run-time instantiation stage, instances of grid workflow *gw* are enacted. We will get absolute start time, i.e. $S(a_1)$. $S(a_1)$ might be different from *Cie(gw)*. As a result, the overall time redundancy $[ftv(F) - S(a_1)] - D(a_1, a_T)$ might be different from that of build-time stage, i.e. $[ftv(F) - Cie(gw)] - D(a_1, a_T)$. Therefore, we need to re-assign the *N* local fixed-time constraints. The specific assigning process is similar to that of build-time stage by replacing *Cie(gw)* with $S(a_1)$. Hence, we simply omit corresponding discussion.

Based on the above discussion, we can derive an algorithm for assigning local fixed-time constraints at build-time stage. The main part of the algorithm is depicted in Algorithm 1.

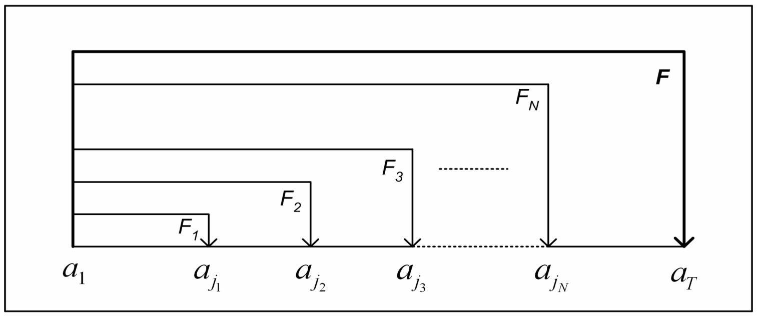


Figure 2. Relationships between *F* and F_1, F_2, ... , F_N

```
Symbol Definitions:
ArrayTA: an array of all $T$ activities covered by the global fixed-time constraint $F$;
ArrayFA: an array of all N activities where we need to set fixed-time constraints;
End Symbol Definitions
Input: ArrayTA, ArrayFA, maximum and mean durations of all activities involved in ArrayTA and ArrayFA;
Output: temporary local fixed-time constraints;
While (not end of ArrayTA)
    // compute all D(a_s) – M(a_s) (s = 1, 2, 3, ... , T)
    Select current activity from ArrayTA to $a_s$ ($s = 1, 2, 3, ... , T$);
    Compute $D(a_s) - M(a_s)$;
End While
Sort all $D(a_s) - M(a_s)$ ($s = 1, 2, 3, ... , T$) in the ascending order to ArrayLA where $D(a_s) - M(a_s)$ is
    ranked No. $k$ in ArrayLA;
While (not end of ArrayTA)
    // allocate time quota to all $T$ activities covered by $F$
    Select current activity from ArrayTA to $a_s$ ($s = 1, 2, 3, ... , T$);
    Apply formula (1) to compute $TQ(a_s)$ as follows;
    $TQ(a_s) = [ftv(F) - Cie(gw) - D(a_1, a_T)] \frac{L_{T-k+1}}{\sum_{l=1}^{T} [D(a_l) - M(a_l)]}$
End While
While (not end of ArrayFA)
    // assign local fixed-time constraints at the $N$ activities respectively
    Select current activity from ArrayFA to $a_{j_i}$ ($i=1, 2, ..., N$);
    Apply formula (2) to compute $ftv(F_i)$ of $F_i$ at $a_{j_i}$ as follows;
    $ftv(F_i) = \sum_{s=1}^{j_i} [TQ(a_s) + D(a_s)]$
End While
```

Algorithm 1. Assigning local fixed-time constraints at build-time stage

4. Verifying and adjusting local fixed-time constraints at run-time execution stage

At run-time execution stage, when grid workflow execution arrives at a checkpoint, say a_p, we verify F_1, F_2, ... , F_N and F. We derive Theorem 1 first which can ease temporal verification.

Theorem 1. At a_p, if a local fixed-time constraint F_s is of SC, then, the consistency of F will not be affected by the execution of a_p.

Proof: According to Definition 3, if F_s is of SC, we have: $R(a_1, a_p) + D(a_{p+1}, a_{j_s}) \leq ftv(F_s)-S(a_1)$. That is to say, the possible time deviation caused by the execution of a_p can be counteracted within F_s, so it will not affect the consistency of F.

Thus, the theorem holds. ▌

In addition, we discussed temporal dependency between fixed-time constraints in [8]. The temporal dependency between fixed-time constraints means that fixed-time constraints are dependent on each other in terms of their verification. We can derive the consistency of later fixed-time constraints from the consistency of previous fixed-time constraints. Correspondingly, we have Theorem 2. The detailed discussion about temporal dependency can be found in [8].

Theorem 2. At a_p, if a local fixed-time constraint F_s is of SC, then, all local fixed-time constraints after F_s must also be of SC.

Proof: Refer to [8].

According to Theorems 1 and 2, we verify F_s firstly. If it is of SC, it means that the time deviation caused by the execution of a_p can be counteracted within F_s. Therefore, we need not verify and adjust any other local fixed-time constraints after F_s as well as the global one, i.e. F.

However, if F_s is not of SC, we need to verify F. There are two situations. One is that F is of SC. This means that the global fixed-time constraint can still be kept even if the temporary local one F_s is violated. In this case, since F_s is set by us to control grid workflow execution rather than from user needs, we do not have to trigger any exception handling to deal with the violation of F_s. We only need to adjust F_s and other remaining local fixed-time constraints based on the current available time redundancy of F. The specific adjustment methods are similar to those assigning ones of build-time stage, hence omitted.

The other situation is that F is violated, i.e. not of SC. In this case, since F_s is also violated, we should try to trigger exception handling to deal with the violation locally within F_s because this will affect fewer activities. Consequently, it would be more cost-effective. If we can handle the violation within F_s, then, according to Theorems 1 and 2, we need not adjust F and other remaining local fixed-time constraints. However, if we cannot handle the violation within F_s, we have to handle it within F. In this case, after we adjust F, we need to adjust all remaining local fixed-time constraints. The corresponding adjustment methods are similar to those assigning ones of build-time stage. The difference is that here we only need to focus on those succeeding unexecuted activities rather than all activities at build-time stage.

Based on the above discussion, we can derive an algorithm for dynamically verifying and adjusting local fixed-time constraints at run-time execution stage within the global fixed-time constraint. However, we simply omit it as it can be derived straightforwardly from the above discussion.

5. Comparison and discussion

In conventional verification work, we always verify the global fixed-time constraint with the consideration of all covered activities. We cannot control the temporal aspect of grid workflow execution in a local range. As a result, for any temporal violations, we must handle them globally encompassing all activities. Comparing with existing related work, the clear difference in this paper is that we have investigated how to assign, verify and adjust local fixed-time constraints. With local fixed-time constraints, we can locally control grid workflow execution. Especially, as stated in Section 4, when temporal violations happen, we can try to handle them locally within local fixed-time constraints rather than within the global one. Local handling will affect fewer activities rather than global handling. So, local handling would be more cost-effective than the global handling. That is to say, we can achieve much better cost-effectiveness with local fixed-time constraints than only with the global fixed-time constraint. In particular, handling of temporal violations is costly, especially in the large-scale grid environments because the handling may touch many different resources spanning large-scale multi-organisations. We need to negotiate with different resource providers with different background to conduct the handling. So, the handling cost is not just from handling itself. It is also partly incurred by the dynamic negotiation process. And sometimes, we may have to compensate some activities or even worse we may need to change the workflow schema locally. In other words, handling cost becomes a more prominent issue in grid environments than in some traditional enterprise environments. Therefore, by nature, it is worth for us to assign local fixed-time constraints towards better cost-effectiveness.

We can further conduct a quantitative analysis to demonstrate the extent to which we can achieve better cost-effectiveness by local fixed-time constraints. We are investigating some simulation and experimental tools or environments to select an appropriate one for further reasoning about the benefits of our work.

6. Conclusions and future work

To control grid workflow execution locally in terms of time for better cost-effectiveness, we have investigated how to assign local fixed-time constraints within the global one. We have developed corresponding assigning methods at build-time and run-time instantiation stages. Then, at run-time execution stage, we have further developed corresponding verification and adjustment methods. The brief comparison and discussion has intuitively shown that we can achieve better cost-effectiveness with local fixed-time constraints than only with the global one.

With the above contributions, we are working on how to facilitate timed Petri-Net for timed grid workflow representation and how to reason about the assigning of local fixed-time constraints based on such representation.

Acknowledgements

The work reported in this paper is partly supported by Australian Research Council Projects under grant No. LP0562500 and grant No. DP0663841.

References

[1] D. Abramson, J. Kommineni, J.L. McGregor and J. Katzfey, “An Atmospheric Sciences Workflow and its Implementation with Web Services”, Future Generation Computer Systems, 2005, 21(1), pp. 69-78.

[2] R. Al-Ali, K. Amin, G.V. Laszewski, O. Rana, D. Walker, M. Hategan, and N. Zaluzec, “Analysis and Provision of QoS for Distributed Grid Applications”, Journal of Grid Computing, 2004, 2(2), pp. 163-182.

[3] G. Aloisio, M. Cafaro, G. Carteni, I. Epicoco, G. Quarta, A. Raolil, "GridFlow for Earth Observation Data Processing", In Proc. of 2005 International Conference on Grid Computing and Applications (GCA 2005), Las Vegas, Nevada, USA, June 2005, pp. 168-176.

[4] I. Brandic, S. Benkner, G. Engelbrecht, R. Schmidt, "Towards Quality of Service Support for Grid Workflows", In Proc. of European Grid Conference 2005 (EGC 2005), Springer-Verlag, Amsterdam, The Netherlands, Feb. 2005, LNCS 3470, pp. 661-670.

[5] R. Buyya, D. Abramson and S. Venugopal, "The Grid Economy", The Proceedings of The IEEE, 2005, 93(3), pp. 698-714.

[6] J. Cao, S.A. Jarvis, S. Saini and G.R. Nudd, "GridFlow: Workflow Management for Grid Computing", In Proc. Of 3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003), IEEE CS Press, Tokyo, Japan, May 2003, pp. 198-205.

[7] D. Cybok, "A Grid Workflow Infrastructure", Concurrency and Computation: Practice and Experience, Special Issue on Grid Workflow, 2006, to appear, http://www.cc-pe.net/CCPEweb resource/c8545to872workflow/c856cybok/c856Grid_Workflow_ Paper_ggfFINAL.pdf, accessed on July 8, 2006.

[8] J. Chen and Y. Yang, "Temporal Dependency for Dynamic Verification of Fixed-date Constraints in Grid Workflow Systems", In Proc. of 7th Asia Pacific Web Conference, Springer-Verlag, Shanghai, China, Mar. 2005, LNCS 3399, pp. 820-831.

[9] 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 (GSEM2005), Erfurt, Germany, Sept. 2005, Lecture Notes in Informatics P-69, pp. 296-310.

[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 12th Asia-Pacific Software Engineering Conference (APSEC2005), IEEE CS Press, Taiwan, Dec. 2005, pp. 299-306.

[11] J. Chen and Y. Yang, "Flexible Temporal Consistency for Fixed-time Constraint Verification in Grid Workflow Systems", In Proc. of 4th International Conference on Grid and Cooperative Computing (GCC2005), Springer-Verlag, Beijing, China, Nov./Dec. 2005, LNCS 3795, pp. 300-311.

[12] J. Chen and Y. Yang, "Multiple States based Temporal Consistency for Dynamic Verification of Fixed-time Constraints in Grid Workflow Systems", Concurrency and Computation: Practice and Experience, 2006, to appear, http://www.it.swin.edu.au/personal/yyang/papers/temporal_states _CCPE.pdf, accessed on July 8, 2006.

[13] J. Chen and Y. Yang, "Selecting Necessary and Sufficient Checkpoints for Dynamic Verification of Fixed-time Constraints in Grid Workflow Systems", In Proc. of 4th International Conference on Business Process Management, Springer-Verlag, 2006, LNCS, to appear, http://www.it.swin.edu.au/personal/ yyang/papers/BPM2006.pdf, accessed on July 8, 2006.

[14] 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.

[15] J. Eder, E. Panagos and M. Rabinovich, "Time Constraints in Workflow Systems", In Proc. of 11th International Conference on Advanced Information Systems Engineering (CAiSE99), Springer-Verlag, Heidelberg, Germany, June 1999, LNCS 1626, pp. 286-300.

[16] T. Fahringer, S. Pllana, A. Villazon, "A-GWL: Abstract Grid Workflow Language", In Proc. of 4th International Conference on Computational Science, Part IV, Springer-Verlag, Krakow, Poland, June 2004, LNCS 3038, pp. 42-49.

[17] I. Foster, C. Kesselman, J. Nick and S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", Technical Report in 5th Global Grid Forum Workshop (GGF5), Edinburgh, Scotland, July 2002, http://www.globus.org/alliance/publications/papers/ogsa.pdf, accessed on July 8, 2006

[18] C. Goble, "Building ad hoc (personal) workflows in an open world: myGrid experiences", Technical Report in 12th Global Grid Forum Workshop (GGF12), Brussels, Belgium, Sept. 2004, http://www.isi.edu/~deelman/wfm-rg/ggf12/GGF12goble.ppt, accessed on July 8, 2006.

[19] Y. Huang, "JISGA: A JINI-BASED Service-Oriented Grid Architecture", The International Journal of High Performance Computing Applications, 2003, 17(3), pp. 317-327.

[20] S. Krishnan, P. Wagstrom and G. von Laszewski, "GSFL: A Workflow Framework for Grid Services", Technical Report, Argonne National Laboratory, Cass Avenue, Argonne, IL 60439, U.S.A., 2002, http://www-unix.globus.org/cog/papers/gsfl-paper.pdf, accessed on July 8, 2006.

[21] J. Li, Y. Fan and M. Zhou, "Timing Constraint Workflow Nets for Workflow Analysis", IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans, 2003, 33(2), pp. 179-193.

[22] 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.

[23] W. Sadiq and M.E. Orlowska, "Analysing Process Models using Graph Reduction Techniques", Information Systems, 2000, 25(2), pp. 117-134.

[24] 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 UK e-Science All Hands Meeting (AHM04), Nottingham, UK, Aug./Sept. 2004, pp. 547-554.

[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.