

Dynamic Setting, Verification and Adjustment of Upper Bound 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

Upper bound constraints are often set when complex scientific or business processes are modelled as grid workflow specifications. However, many existing processes such as climate modelling or international stock market analysis often have only one end-to-end upper bound constraint. This is not sufficient to control overall temporal correctness as we may not find temporal violations until the last activity. Then, it is too late to take any handling actions. Consequently, the execution results may not be useful and overall cost-effectiveness would be impacted. Therefore, in this paper, we systematically investigate how to set, verify and adjust sub-upper bound constraints within the timeframe of one end-to-end upper bound constraint so that we can control grid workflow execution locally. We develop corresponding setting, verification and adjustment methods and algorithms. The quantitative evaluation demonstrates that with sub-upper bound constraints, we can achieve better cost-effectiveness than only based on one end-to-end upper bound constraint.

1. Introduction and motivation

In the grid architecture, a grid workflow system is a high-level grid middleware. It is supposed to support modelling, redesign and execution of large-scale sophisticated e-science and e-business processes in many complex scientific and business applications such as climate modelling, astrophysics, international finance and insurance [1, 3, 17, 22]. At build-time stage, complex scientific or business processes are modelled or redesigned as grid workflow specifications by some grid workflow definition languages [6, 16, 20]. According to [6], conceptually, a grid workflow contains a large number of computation, data or transaction intensive activities, and dependencies between them. These activities are implemented and executed by corresponding grid services [6, 9, 18, 20]. At run-time instantiation stage, grid

workflow instances are created, and especially grid services which are specified in build-time definition documents are discovered by a high-level instantiation grid service [6, 9, 20]. At run-time execution stage, grid workflow instances are executed, which is coordinated by a high-level workflow engine grid service [6, 9, 20].

In reality, complex scientific or business processes are normally time constrained. Consequently, upper bound constraints are often set [15, 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 at build-time, run-time instantiation and execution stages to check if all upper bound constraints are consistent.

In many scientific and business processes such as climate modelling, international stock market analysis, we often have only one end-to-end upper bound constraint [1, 5]. From the perspective of user needs, only one end-to-end upper bound constraint is intuitive and simple. However, from the perspective of specific grid workflow execution, it is not sufficient to ensure overall temporal correctness. This is because we cannot control local temporal correctness. As a result, we may only find out the temporal violation until the last activity. Then, it is too late to take any handling actions. Consequently, the execution results will not be useful and the overall cost-effectiveness will be severely impacted. Therefore, we must investigate how to set, verify and adjust sub-upper bound constraints within the end-to-end upper bound constraint so that we can control grid workflow execution locally. However, existing conventional temporal verification work does not pay sufficient attention to this issue. Hence, in this paper, we are making an effort to fill the gap. We systematically investigate how to set, verify and adjust sub-upper bound constraints. Based on the investigation, we develop corresponding setting, verification and adjustment methods and algorithms. With sub-upper bound constraints, we can achieve much better cost-effectiveness than only with one end-to-end upper bound constraint. The quantitative evaluation further demonstrates this result.

Specifically, in Section 2, we summarise related work. In Section 3, we represent some time attributes of grid workflows. In Section 4, we discuss how to set, verify and adjust sub-upper bound constraints. We also develop some setting, verification and adjustment algorithms in Section 4. In Section 5, we conduct a quantitative evaluation which demonstrates that based on these algorithms, we can achieve much better cost-effectiveness than only based on one end-to-end upper bound constraint. Finally in Section 6, we conclude our contributions and point out future work.

2. Related work

According to the literature, [2, 4] analyses QoS (Quality of Service) including temporal QoS in distributed grid (workflow) applications and discusses how to provide QoS. [5] discusses grid economy issues including temporal aspects in grid architecture. [9] investigates multiple temporal consistency states for upper bound constraints in grid workflow systems. [15] uses the modified Critical Path Method (CPM) to calculate temporal constraints. [19] investigates exception handling in workflow management systems. [21] presents a method for dynamic verification of absolute deadline constraints and relative deadline constraints. [23] proposes a timed workflow process model with considering the flow time, the time difference in a distributed execution environment. [24, 25] discuss knowledge grid which is a new paradigm for promising research on knowledge and grid.

The above related work has presented some background for temporal aspect in grid workflows such as some timed grid workflow models or temporal verification methods. However, they do not pay sufficient attention to how to set, verify and adjust sub-upper bound constraints within the timeframe of one end-to-end upper bound constraint. In this paper, we are making an effort to fill this gap.

3. Timed grid workflow representation

According to [13, 15], 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 them. To represent time attributes in a grid workflow, we borrow some concepts from [15, 21] such as maximum, mean or minimum duration as a basis. We denote the i^{th} activity of a grid workflow as a_i and 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 $Rcd(a_i)$ respectively. $M(a_i)$ means that statistically a_i can be completed around its mean duration. Other time attributes are self-explanatory. According to [15, 21], $D(a_i)$, $M(a_i)$ and $d(a_i)$ can be obtained based on the past execution history which covers the delay time incurred at a_i such as setup delay, queuing

delay, synchronisation delay, network latency and so on. The detailed discussion on how to obtain and set $D(a_i)$, $M(a_i)$ and $d(a_i)$ is outside the scope of this paper and can be found in [15, 21]. For a specific execution of a_i , the delay time is included in $Rcd(a_i)$. Normally, we have $d(a_i) \leq M(a_i) \leq D(a_i)$ and $d(a_i) \leq Rcd(a_i) \leq D(a_i)$.

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 [15, 21]. If there is an upper bound constraint between a_i and a_j , we denote it as $UBC(a_i, a_j)$ and 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 to the end, it is still an execution path. Therefore, for the selective/parallel/iterative structures, we can also apply the results achieved from one execution path.

Besides the above time attributes, four temporal consistency states have been identified and defined in [9, 11] which 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 [9, 11].

Definition 1. 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 2. At run-time instantiation 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 checkpoint¹ 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 $ubv(a_i, a_j) < Rcd(a_i, a_p) + d(a_{p+1}, a_j)$.

In this paper, we focus on SC only. The corresponding discussion for WC, WI and SI is similar.

4. Dynamic setting, verification and adjustment of sub-upper bound constraints

4.1. Build-time stage

4.1.1. Setting. We denote the end-to-end upper bound constraint as U and its value as $upv(U)$. Based on the past execution history, we can summarise those grid workflow

¹ A checkpoint is an activity point where temporal verification must be conducted. Refer to [7, 8, 10, 12, 22, 25] for more details.

path slots where temporal violations often happen. Accordingly, at each slot we should set a sub-upper bound constraint. Suppose there are N such slots. Then, we need to set N sub-upper bound constraints. We denote them as U_1, U_2, \dots , and U_N , and their values as $ubv(U_1), ubv(U_2), \dots$, and $ubv(U_N)$. We suppose U_i covers M_i activities, denoted as a_{ij} ($j=1, 2, 3, \dots, M_i$). In addition, we suppose there are T activities covered by U . Then, if U is of SC, we have a time redundancy: $ubv(U) - D(a_1, a_T)$ which can be used to tolerate certain time deviation. Based on this time redundancy, we can derive U_1, U_2, \dots, U_N as follows.

Suppose there are totally M activities covered by U_1, U_2, \dots, U_N , i.e. $M_1+M_2+M_3+\dots+M_N = M$. Among M activities, we sort all $D(a_{ij}) - M(a_{ij})$ ($i=1, 2, 3, \dots, N; j=1, 2, 3, \dots, M_i$) in ascending order and get a sorting list. We denote the list as L and the items in L as L_1, L_2, \dots, L_M . If $D(a_{ij}) - M(a_{ij})$ is ranked No. k in L , i.e. L_k , we propose (1) to allocate $ubv(U) - D(a_1, a_T)$ to each of the M activities. We denote the time quota allocated to a_{ij} as $TQ(a_{ij})$.

$$TQ(a_{ij}) = [ubv(U) - D(a_1, a_T)] \frac{L_{M-k+1}}{\sum_{l=1}^M [D(a_l) - M(a_l)]} \quad (1 \leq k \leq M)$$

The relationship between L_k and L_{M-k+1} is depicted in Figure 1.

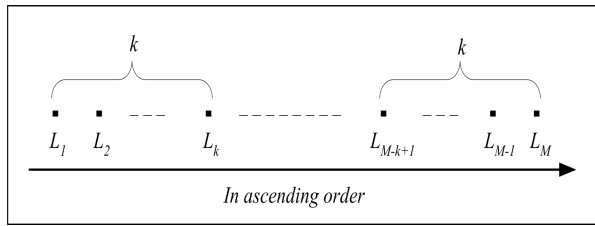


Figure 1. Relationship between L_k and L_{M-k+1}

In (1), we allocate $ubv(U) - D(a_1, a_T)$ to activities covered by U_1, U_2, \dots , and U_N based on the difference between the activity maximum duration and the activity mean duration. The activity with a bigger difference will be allocated a smaller quota of $ubv(U) - D(a_1, a_T)$. 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 $ubv(U) - D(a_1, a_T)$, we can derive new U_1, U_2, \dots , and U_N . For U_i , we propose (2).

$$ubv(U_i) = \sum_{j=1}^{M_i} [TQ(a_{ij}) + D(a_{ij})] \quad (i=1, 2, 3, \dots, N) \quad (2)$$

The basic relationship between U and U_1, U_2, \dots , and U_N is shown in Figure 2.

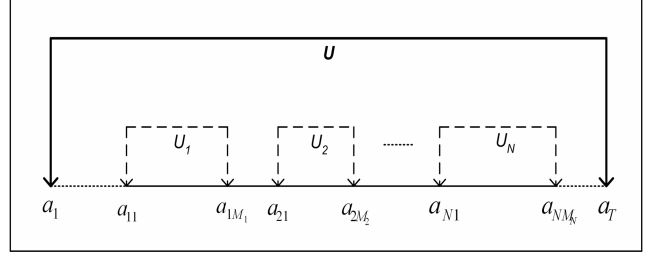


Figure 2. Relationship between U and U_1, U_2, \dots , and U_N

4.1.2. Verification and adjustment. We discuss verification and adjustment in the same section as they are interleaved. To verify U_1, U_2, \dots, U_N and U , we derive Theorem 1.

Theorem 1. Given an end-to-end upper bound constraint U , if we allocate its time redundancy $ubv(U) - D(a_1, a_T)$ according to formula (1) and derive sub-upper bound constraints according to formula (2), then, if U is of SC, all sub-upper bound constraints are also of SC.

Proof: Considering a sub-upper bound constraint, say U_i . If U is of SC, $ubv(U) - D(a_1, a_T) > 0$. Then, $TQ(a_{ij}) > 0$ ($j=1, 2, 3, \dots, M_i$). Hence, with formula (2), we have:

$$ubv(U_i) = \sum_{j=1}^{M_i} [TQ(a_{ij}) + D(a_{ij})] > \sum_{j=1}^{M_i} D(a_{ij}) = D(a_{i1}, a_{iM_i})$$

According to Definition 1, this means that U_i is of SC. Thus, the theorem holds. ■

According to Theorem 1, when we verify U_1, U_2, \dots, U_N and U , we first verify U . If U is of SC, we need not verify U_1, U_2, \dots , and U_N . If U is not of SC, we need to re-set U . For example, we enlarge U so that it can be of SC. However, once we do so, we need to adjust all sub-upper bound constraints. Correspondingly, we need to conduct the allocating and deriving process in Section 4.1.1 again to achieve new values of U_1, U_2, \dots , and U_N .

4.2. Run-time instantiation stage

At run-time instantiation stage, grid workflow instances are enacted and we will get their start time, i.e. $S(a_i)$. However, we do not have specific activity execution times. So, if there are no any new activities added to grid workflow specifications, we need not conduct any setting, verification and adjustment but simply keep corresponding build-time results. Otherwise, we need to re-set, re-verify and re-adjust the sub-upper bound constraints and the end-to-end one, i.e. U_1, U_2, \dots, U_N and U . Since $S(a_i)$ is the only different time attribute between build-time stage and run-time instantiation stage and the consistency of U_1, U_2, \dots, U_N and U has nothing to do with $S(a_i)$, the corresponding re-setting, re-verification and re-adjustment are similar to those of build-time stage, hence omitted.

4.3. Run-time execution stage

4.3.1. Re-setting. At run-time execution stage, some temporary activities such as temporary data transfer activities might be added to grid workflow instances [9, 14]. If so, we need to adjust the end-to-end upper bound constraint and consequently need to re-set sub-upper bound constraints. The corresponding re-setting method and process are similar to those used at build-time stage, hence omitted.

4.3.2. Verification and adjustment. When grid workflow execution arrives at a checkpoint, say a_p , we verify U_1, U_2, \dots, U_N , and U . We firstly derive Theorem 2.

Theorem 2. At checkpoint a_p which is covered by sub-upper bound constraint U_k , if $Rcd(a_p) \leq D(a_p)$, the execution of a_p does not affect the consistency of U_k and U .

Proof: Suppose U_k is of SC before the execution of a_p . In fact, if U_k is not of SC, we should call some exception handling to adjust it to SC in the first place. In other words, the grid workflow execution would not have arrived at a_p . Then, according to Definition 3, we have: $Rcd(a_{k1}, a_{p-1}) + D(a_p, a_{kM_k}) \leq ubv(U_k)$. If $Rcd(a_p) \leq D(a_p)$, we have: $Rcd(a_{k1}, a_p) + D(a_{p+1}, a_{kM_k}) = Rcd(a_{k1}, a_{p-1}) + Rcd(a_p) + D(a_p, a_{kM_k}) \leq Rcd(a_{k1}, a_{p-1}) + D(a_p) + D(a_p, a_{kM_k}) = Rcd(a_{k1}, a_{p-1}) + D(a_p, a_{kM_k}) \leq ubv(U_k)$. Namely, $Rcd(a_{k1}, a_p) + D(a_{p+1}, a_{kM_k}) + D(a_p, a_{kM_k}) \leq ubv(U_k)$. According to Definition 3, U_k is still of SC after the execution of a_p . Similarly, we can also prove that U is also of SC. That is to say, the execution of a_p does not affect the consistency of U_k and U .

Thus, the theorem holds. ■

According to Theorem 2, at a_p , if $Rcd(a_p) \leq D(a_p)$, we need not verify U and U_k . As to other sub-upper bound constraints, since they do not cover a_p , the execution of a_p does not affect their consistency either. Therefore, in overall terms, if $Rcd(a_p) \leq D(a_p)$, we need not conduct any temporal verification. However, we can get an extra time redundancy, i.e. $D(a_p) - Rcd(a_p)$. We can allocate it to the remaining sub-upper bound constraints, i.e. U_k, U_{k+1}, \dots , and U_N so that they have more time redundancy to tolerate possible time deviation of grid workflow execution. We allocate $D(a_p) - Rcd(a_p)$ to unexecuted activities covered by U_k, U_{k+1}, \dots , and U_N . To do so, we get a sub set of L . L consists of the unexecuted activities covered by U_k, U_{k+1}, \dots , and U_N . Suppose there are totally R activities covered by U_k, U_{k+1}, \dots , and U_N , i.e. $M_k + M_{k+1} + \dots + M_N = R$. Considering an activity among the R activities, say a_{rs} ($r = k, k+1, \dots, N; s = 1, 2, 3, \dots, M_r$). If $D(a_{rs}) - M(a_{rs})$ is ranked No. t in the sub set, i.e. L_t , then we propose (3) to allocate $D(a_p) - Rcd(a_p)$ to each of the R activities. We denote the extra quota allocated to a_{rs} as $EQ(a_{rs})$.

$$EQ(a_{rs}) = [D(a_p) - Rcd(a_p)] \frac{L_{R-t+1}}{\sum_{l=1}^R [D(a_l) - M(a_l)]} \quad (1 \leq t \leq R) \quad (3)$$

The relationship between L_t and L_{R-t+1} is similar to that between L_k and L_{M-k+1} as shown in Figure 1.

After we allocate $D(a_p) - Rcd(a_p)$ to unexecuted activities covered by U_k, U_{k+1}, \dots , and U_N . We can derive new U_k, U_{k+1}, \dots , and U_N . Considering U_j ($j = k, k+1, \dots, N$), we derive its new value by (4) below.

$$ubv(U_j) = ubv(U_j) + \sum_{s=1}^{M_j} EQ(a_{js}) \quad (j = k, k+1, \dots, N) \quad (4)$$

If $Rcd(a_p) > D(a_p)$, we need to verify U_k and even U . To do so, we firstly derive Theorem 3.

Theorem 3. At a_p , given $Rcd(a_p) > D(a_p)$, if U_k is still of SC, the execution of a_p does not affect the consistency of U .

Proof: According to Definition 3, if U_k is of SC, we have: $Rcd(a_{k1}, a_{p-1}) + D(a_p, a_{kM_k}) \leq ubv(U_k)$. That is to say, the time deviation $Rcd(a_p) - D(a_p)$ caused by the execution of a_p can be counteracted within U_k , so it will not affect the consistency of U .

Thus, the theorem holds. ■

According to Theorem 3, if $Rcd(a_p) > D(a_p)$, we verify U_k firstly. If it is of SC, we need not verify U . Since other sub-upper bound constraints do not cover a_p , we need not verify them either. That is to say, if U_k is of SC, we need not conduct any other temporal verification. In addition, if U_k is of SC, it means that the time deviation caused by the execution of a_p can be counteracted within U_k . Therefore, we need not adjust other sub-upper bound constraints and the end-to-end one.

However, if $Rcd(a_p) > D(a_p)$ and U_k is not of SC. We need to verify U . There are two situations. One is that U is of SC. This means that the end-to-end upper bound constraint can still be kept even if U_k is violated. In this case, since U_k 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 U_k . We only need to adjust U_k as well as other remaining sub-upper bound constraints based on U again given that the available time redundancy of U has changed. The specific adjustment methods are similar to those setting ones in Section 4.1.1, hence omitted.

The other situation is that U is violated, i.e. not of SC. In this case, since U_k is also violated, we firstly should try to deal with the violation locally within U_k because this would affect fewer activities. Consequently, it would be more cost-effective. If the violation can be handled within U_k , then, according to Theorem 3, we need not adjust U further. Correspondingly, we need not adjust other remaining sub-upper bound constraints either. However, if we cannot handle the violation within U_k , we must handle it within U . In this case, after we adjust U , we need to

adjust the remaining sub-upper bound constraints. The corresponding adjustment methods are similar to those setting ones in Section 4.1.1. The difference is that here we only need to focus on those succeeding unexecuted activities rather than all activities in Section 4.1.1.

Based on the above discussion, we can derive an algorithm for dynamically re-setting, verifying and adjusting sub-upper bound constraints at run-time execution stage. However, due to the page limit, we simply omit it.

5. Comparison and quantitative evaluation

Comparing with existing related work, the clear difference in this paper is that we have systematically investigated how to set, verify and adjust sub-upper bound constraints. With sub-upper bound constraints, according to Section 4.3.2, we can try to handle temporal violations locally within sub-upper bound constraints rather than within the end-to-end upper bound constraint. This would be more cost-effective as local handling affects fewer activities. We can conduct a quantitative analysis of how the introduction of sub-upper bound constraints can achieve better cost effectiveness. However, due to the page limit, we only describe the quantitative analysis briefly.

We use some symbols. S : number of sub-upper bound constraints. X : number of temporal violations which can be handled within a sub-upper bound constraint (we suppose that X temporal violations happen respectively at the first X activities of each sub-upper bound constraint). Q : number of activities between any two adjacent sub-upper bound constraints. P : number of activities covered by each sub-upper bound constraint. C : exception handling cost at an activity. $DIFF$: exception handling cost based on one end-to-end upper bound constraint minus that based on sub-upper bound constraints.

Then, we can derive (5).

$$DIFF = \sum_{i=1}^S \sum_{k=1}^X [i*Q+(i-1)*P+k]*C - S*\sum_{k=1}^X k*C \quad (5)$$

By taking a set of specific values, we depict how $DIFF$ changes to S in Figure 3. We suppose that $P=3$, $Q=2$, $X=2$, C is equal to 1 cost unit and S can change from 0 to 20. The selection of these specific values is random and does not affect our analysis because what we want to see is the trend of how $DIFF$ changes to S .

According to Figure 3, we can see: when S is getting larger, $DIFF$ is getting much larger. Since in real-world grid workflow systems, grid workflows are normally very complicated and normally last a long time [1, 22]. Therefore, to better control local grid workflow execution, a good number of sub-upper bound constraints are often needed, i.e. S is normally a large number. Therefore, in overall terms, with multiple sub-upper bound constraints, we can achieve much better cost-effectiveness.

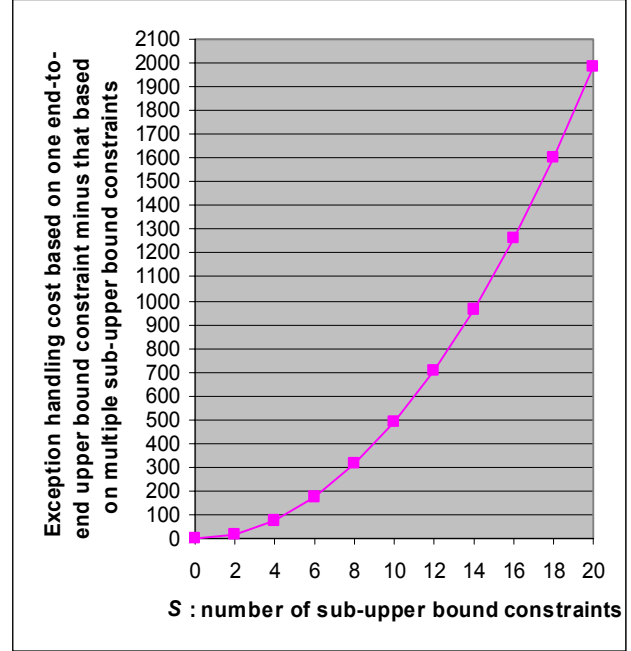


Figure 3. Change trend of $DIFF$ by number of sub-upper bound constraints - S

6. Conclusions and future work

Many scientific and business processes have only one end-to-end upper bound constraint. This is not sufficient to ensure the overall temporal correctness of grid workflow execution because we may not find the temporal violation until the last activity. Then, it is too late to take any handling actions. As a result, the overall cost-effectiveness would be impacted. Therefore, in this paper, we have systematically investigated how to set, verify and adjust sub-upper bound constraints within the timeframe of one end-to-end upper bound constraint. We have also developed corresponding setting, verification and adjustment methods and algorithms. The quantitative evaluation has shown that with sub-upper bound constraints, we can achieve much better cost-effectiveness than only with the end-to-end upper bound constraint.

With these contributions, we can further investigate temporal exception handling approaches when a sub-upper bound constraint is violated. Another future work is to investigate how to incorporate the theoretical results of this paper into semantic or knowledge grid based on [24, 25] such as semantic grid workflow or knowledge flow.

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] 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, Hawaii, Jan. 2004, pp. 210-219.
- [4] I. Brandic, S. Benkner, G. Engelbrecht, R. Schmidt, "Towards Quality of Service Support for Grid Workflows", In Proc. of European Grid Conference 2005 (EGC2005), Amsterdam, The Netherlands, Feb. 2005.
- [5] R. Buyya, D. Abramson, and S. Venugopal, "The Grid Economy", *Proceedings of The IEEE*, 2005, 93(3), pp. 698-714.
- [6] D. Cybok, "A Grid Workflow Infrastructure", In Proc. of Workflow in Grid Systems Workshop in GGF10, Berlin, Germany, Mar. 2004.
- [7] 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, IEEE CS Press, Busan, Korea, Nov./Dec. 2004, pp. 30-37.
- [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 the 2nd International Conference on Grid Service Engineering and Management, Sept. 2005, LNI P-69, pp. 296-310.
- [9] J. Chen, and Y. Yang, "Multiple Temporal Consistency States for Dynamical Verification of Upper Bound Constraints in Grid Workflow Systems", In Proc. of the 1st International Conference on e-Science and Grid Computing, Dec. 2005, pp. 124-131.
- [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, pp. 299-306.
- [11] 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/yayang/papers/temporal_states_CCPE.pdf.
- [12] 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/yayang/papers/BPM2006.pdf>, accessed on July 8, 2006.
- [13] 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.
- [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 the 11th International Conference on Advanced Information Systems Engineering (CAiSE'99), 1999, LNCS 1626, pp. 286-300.
- [16] T. Fahringer, S. Pllana, 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.
- [17] 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, <http://www.globus.org/research/papers/ogsa.pdf>, accessed on July 8, 2006.
- [18] C. Goble. Building ad hoc (personal) workflows in an open world: myGrid experiences. In Proc. of the 12th Global Grid Forum Workshop, Brussels, Belgium, Sept. 2004.
- [19] C. Hagen, and G. Alonso, "Exception Handling in Workflow Management Systems", *IEEE Transactions on Software Engineering*, 2000, 26(10), pp. 943-958.
- [20] 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.
- [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, 2004, pp. 547-554.
- [23] H. Zhuge, T. Cheung, and H. Pung, "A Timed Workflow Process Model", *The Journal of Systems and Software*, 2001, 55(3), pp. 231-243.
- [24] H. Zhuge, *The Knowledge Grid*, World Scientific Publishing Co., Singapore, 2004.
- [25] H. Zhuge, "China's E-Science Knowledge Grid Environment", *IEEE Intelligent Systems*, 2004, 19 (1), pp. 13-17.