GRID WORKFLOW SCHEDULING WITH TEMPORAL DECOMPOSITION *

Fei Long and Hung Keng Pung
Network Systems and Services Lab., Department of Computer Science, National University of Singapore, Singapore

Keywords: Grid Workflow, Scheduling, Decomposition, QoS.

Abstract: Grid workflow scheduling, a very important system function in current Grid Systems, is known as a NP complete problem. In this paper, we propose a new scheduling method– "temporal decomposition" – which divides a whole grid workflow into some sub-workflows. By dividing a large problem (workflow) into some smaller problems (sub-workflows), the "temporal decomposition" to better exploit achieves much lower computation complexity. Another motivation for the design of "temporal decomposition" is at the availability of the highly dynamic grid resources. We further propose an efficient scheduling algorithm for scheduling sub-workflows in this paper. Numerical simulation results show that our proposed scheme is more efficient in comparison with a well known existing grid workflow scheduling method.

1 INTRODUCTION

The grid workflow has some unique characteristics which are different from conventional workflow, such as business workflow(Zhang et al., 2004). These characteristics include: 1) Highly dynamic grid resources. Unlike in traditional workflow systems, resources in grid network can join and leave the network at any time, e.g. during the execution of associated grid tasks. Such changes of resource quantity may lead to cause the failure of task execution or even locked situation. 2) Distributed characteristic. In conventional workflow system, the resources are concentrative and managed in centrally concentrative way. However, both resources and resource management system (RMS) are distributed in current grid networks. Thus it is impossible to obtain global resources information for a single request. 3) Uncertain execution time. Due to the variance in assigned resource, the execution time of each grid task is uncertain. Furthermore, the capacity of each resource in the grid network is highly diverse, which makes the execution time different even for the same task assigned to different resources. Due to these factors, the conventional workflow management system can't be applied to grid environment directly. A new grid workflow scheduling algorithm is clearly desirable. In this paper, we propose a new distributed workflow scheduling algorithm for p2p grid network.

It is well known that the scheduling of grid workflow to distributed grid resources is a NP-complete problem. One effective way to solve such problem is to divide a big problem into some smaller subproblems. Thus decomposition of workflow is meaningful for solving grid workflow scheduling problem. Besides space decomposition(Yu et al., 2005) of workflow, which divides the workflow into some parts according to the workflow structure and relationships between tasks, in this paper we present another method of dividing grid workflow into smaller parts; it is known as temporal decomposition. Temporal decomposition divides workflow according to the estimated start time of a task and its execution dependencies with other between tasks. Suppose the scheduler is scheduling tasks starting from time t_0. Only the tasks whose preceeding tasks have all finished or running, and whose start times are less than the $t_0 +$ "scheduling window"(an important system parameter) will be considered in current schedul-

*The work is funded by SERC of A*Star Singapore through the National Grid Office (NGO) under the research grant 0520150024 for two years.

Long F. and Keng Pung H. (2007).
GRID WORKFLOW SCHEDULING WITH TEMPORAL DECOMPOSITION.
In *Proceedings of the Ninth International Conference on Enterprise Information Systems - ISAS*, pages 441-446
DOI: 10.5220/0002373904410446

ing decision.

Our main contributions in this paper are two folders. First, we use temporal decomposition to reduce the complicated grid workflow scheduling problem to simplified the scheduling of sub-workflow (parallel tasks), which is easier to solve and realized. Secondly, we propose an efficient on-line scheduling algorithm for the decomposed sub-workflows. By using a QoS bidding mechanism, this scheduling algorithm can find the near-minimum cost decision without violations of QoS constraints.

This paper is organized as follows. In Section 2, we present some important existing grid workflow scheduling heuristics. Section 3 presents the system model used in this work. Our proposal is introduced in Section 4. Section 5 presents some simulation results which demonstrate the effectiveness of our proposal. Finally, we conclude this work paper in Section 6.

2 WORKFLOW DECOMPOSITION

One possible method for solving NP-complete problem is to divide a big problem into some small problems, since the computation complexity decreases dramatically with the problem size. Thus the approach of decomposition of workflow structure is an attractive approach for solving grid workflow scheduling problem. There are two different approaches to de-compose the workflow structure– "space decomposition" and "level decomposition"(Deelman et al., 2004). "Space decomposition" divides the workflow into some parts according to the workflow structure and relationships between tasks. For example, Yu(Yu et al., 2005) divides workflow into independent branches and synchronization tasks, and schedules these branches or synchronization tasks separately. A synchronization task is defined as a task with multiple preceding tasks or succeeding tasks, while the task with only one or less preceding task and succeeding task is called simple task. A branch contains a series of simple tasks executed sequentially between two synchronization tasks. The decomposition result of "space decomposition" is highly dependent on the workflow structure. For example, there are many synchronization tasks in a workflow with small number of serial tasks. The scheduling for synchronization task is far from optimal since only one task has been considered.

A simple "level decomposition" method has been proposed in(Deelman et al., 2004) . In such decomposition method, the abstract workflow is decomposed into some sub-workflows, which consist of tasks with the same level (determined by the execution dependency) in the abstract workflow structure. The new sub-workflows will be submitted to a scheduler, and the scheduler will make decision based on all tasks in a sub-workflow together instead of individual task. "Level decomposition" is too simple to support complicated workflow components, such as loop task. Another shortcoming of "level decomposition" is that the next-level sub-workflow will not start to execute until all tasks in previous-level sub-workflow have been completed.

3 SYSTEM MODELS

In this paper, we adopt the definition of grid workflow as an abstract representation of application running on grid networks. Grid workflow is a set of tasks that are executed on multi-sited grid resources in a specific pre-defined order(Yu and Buyya, 2005). Grid tasks are the atomic level components in the grid workflow. These components are independently executed on local grid resources. The abstract grid tasks may represent various application components, such as MPI tasks which can execute on multiple processors. Thus, these tasks have various QoS requirements, which could be satisfied by choosing good schedules with appropriate resource mapping. The typical QoS metrics used in grid networks includes(Cardoso et al., 2004): time (deadline), cost (budget), and reliability. As the basic and most important performance metric, "time" refers to the finish time of whole workflow execution. The usage of grid resources will be charged by the resource owner, if the resource does not belong to the user submitting the workflow. Furthermore, the cost of managing workflow in grid system should also be paid born by the users. The execution of grid tasks is depend on the availability and reliability of the resources. For example, when a resource leaves the grid system, all tasks running on it will fail and should be re-executed on another resource.

3.1 Qos Bid Model

Suppose there are M distributed resources in the system. All resources support resource conservation with a limited amount of processing capacity. The capacities of resources are represented by $\{c_i\}, i = 1, \ldots, M$, which may have different definitions (e.g. number of processors, CPU cycles, memory size). These resources are shared among the end-users with different QoS requirements. There is a local agent (LA) for each resource as shown in Figure 1.

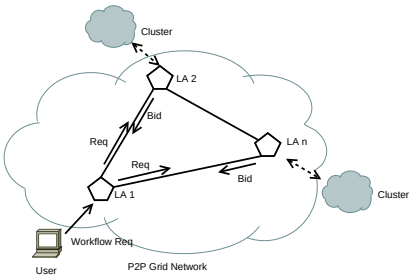

Figure 1: Grid Framework.

Each LA can play one of two roles– as a scheduler or a contractor. A scheduler LA collects the workflow requests from end-users, and is responsible for scheduling the workflow to distributed resources. The contractor LA is referred to the LA which accepts task from the scheduler LA and performs local scheduling, runtime state monitoring and etc. The roles of LAs are not fixed and are not pre-defined in advance. In fact, these roles are dynamically changed by the requirement of users. Therefore, a LA may alternate between two roles or maintain both roles in the same time during the system operation.

When a workflow request arrives at the scheduler LA, the LA will at first analyze the structure of workflow and de-composite the workflow into tasks. The individual QoS requirements of each task will be derived from the QoS requirement of whole workflow. Each task has its own QoS requirement and capacity requirement, represented by q_j and r_j $(j = 1,\ldots,N)$ respectively. The QoS requirement typically has a tolerance bound, which is presented by a percentage value b_j. In other words, the bound for QoS requirement is $q_j(1 \pm b_j)$. Then the LA will query the available resources by broadcast the query message to neighborhood contractor LAs, which can receive the broadcast message. After receiving the query message, the neighborhood LAs will check their available resources capacity and compare it to the QoS requirement of the task. If the QoS requirement of the task can be satisfied, the contract LA will quote a price and send back its bid to the scheduler LA. The query message contains two important time information– deadline of this task and life time of this query message; the latter indicates the longest resource query time for this task. In other words, the contractor LA should reply within the life time, otherwise the reply will be viewed as expired.

The QoS bids for one task will be stored in an individual bid queue. All tasks have their own bid queues. The order of bids in the queue can be adjusted by the scheduler. The scheduler will choose a bid for each task in the sub-workflow. The set of selected bids for the sub-workflow constitutes the schedule decision.

3.2 Workflow Model

The previous work on grid workflow model can be classified to two types– *abstract* and *concrete*. In the *abstract* model, grid workflow is defined without referring to specific grid resources. In the *concrete* model, the definition of workflow includes both workflow specification and related grid resources. Defining a new model for scientific grid application is beyond the scope of this paper. We adopt the abstract DAG model to describe grid workflow as following definition.

Definition 1 *A grid workflow is a DAG denoted by* $\mathcal{W} = \{\mathcal{N}, \mathcal{E}\}$*, where* $\mathcal{N}$ *is the set of grid tasks and* $\mathcal{E}$ *the set of directed task dependencies. Let* $s(n)$, $p(n)$ *be the sets of succeeding tasks and preceding tasks of task* $n \in \mathcal{N}$ *respectively.* $n_s \in s(n)$ *means* $\exists(n, n_s) \in \mathcal{E}$*, while* $n_p \in p(n) \iff \exists(n_p, n) \in \mathcal{E}$*.*

In this model, the grid task is an abstract definition, which can be the representation of various categories of tasks, such as data transfer task and calculation task and etc.

3.3 Optimal Objective

Price function $P(i, j)$(Smale, 1976)(Wolski et al., 2001)(Cheng and Wellman, 1998) represents the cost of executing task j at resource i. Thus, our design objective is to minimize the total cost of workflows.

$$minimum \sum P(i, j) \tag{1}$$

subject to,

$$\begin{array}{l} q_j \text{ is satisfied} \forall j \\ \sum_j r_j x_{i,j} < c_i \forall i, j \end{array} \tag{2}$$

where

$$x_{i,j} = \begin{cases} 1, \text{task j is assigned to resource i} \\ 0, \text{else.} \end{cases} \tag{3}$$

4 PROPOSAL

4.1 Maximum Parallel Tasks

Unlike traditional static scheduling for the whole grid workflow, we introduce the concept of "Maximum parallel tasks" (MPT) for decomposing workflow. Workflow decomposition provides a feasible alternative way for reducing a large workflow scheduling problem to some smaller sub-workflow scheduling problems. Besides space decomposition, there is another method of dividing grid workflow into smaller

parts– temporal decomposition. Temporal decomposition divides workflow according to the estimated start/end time of tasks.

Another idea behind "MPT" is to reduce workflow scheduling problem to parallel tasks (sub-workflow) scheduling. Different from independent parallel tasks, tasks of grid workflow have some inter-dependency, such as relationship of execution order and relevancy of data. These inter-dependencies make the workflow scheduling more complicated, since the schedule should satisfy these dependency constraints. Dividing the whole workflow into sets of independent parallel tasks will convert a complex workflow scheduling problem to a simpler parallel tasks scheduling problem.

We use MPT to divide the whole workflow into some parts. Thus the scheduling problem complexity is highly reduced due to the smaller size of MPT compared to that of whole workflow. MPT is defined as the waiting tasks which have no un-started preceding tasks (i.e. all their preceding tasks have been completed or are running), and are within next scheduling window. For example, currently there are three parallel tasks (task T1, T2 and T3) running in the system, as shown in Figure 2. Task T4, T5 and T6 are the succeeding task of task T1, T2 and T3, respectively. Suppose task T1 finishes before task T2 and T3, when task T1 finishes, the MPT task set will be task T4, T5 and T6, because the preceeding tasks (T2 and T3) of T5 and T6 are running and the estimated finish times of T2 and T3 do not exceed the scheduling window. Since task T2 and T3 are still running, task T5 and T6 will be scheduled to start after the estimated finish time of their preceding tasks. The big advantage of grid computing is its parallel computation capability. Thus we argue a grid workflow should contain a large proportion of parallel tasks in its structure, in order to utilize the parallel computation capability in grid scenario. Therefore, the size of MPT should not be too small (e.g. equals to 1). Furthermore, usually there are multiple workflow requests at the same LA.

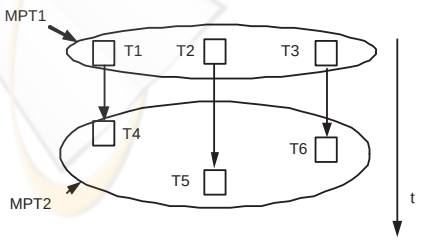

Figure 2: Example of maximum parallel tasks.

Scheduling window is an important design parameter for our system. The length of scheduling window is the space between current scheduling time and next scheduling time. It directly determines the size of MPT waiting to be scheduled. In other words, the value of scheduling window is proportional with the size of MPT. Therefore, its value should neither be too small nor be too great. The selection of next scheduling time point should consider this aspect. The selection algorithm for the next scheduling time is shown in Algorithm 1.

Algorithm 1 Next scheduling time algorithm.

$\{f_i\} \leftarrow$ finish times of tasks with unscheduled succeeding task(s) in current MPT
$t_{min} \leftarrow$ the task with earliest finish time $f_{min} = \min\{f_i\}$
next scheduling time $\leftarrow t_{min}$

Can MPT deal with more complex Grid Workflow components, such as split, merge, condition, loop task? Split component is a Grid task with multiple succeeding tasks. Split component is fully supported by MPT. When the split task is running or finished, its child tasks (if they have no other preceeding task) can be included in the next MPT and wait for scheduling. As the opposite of split component, merge component is a Grid task with multiple preceeding tasks. It can also be supported by MPT if we introduce the following constraint: unless all the preceeding tasks of a merge task are running or finished, this merge task can not be scheduled. However, this constraint may impair the performance of scheduler, since fewer number of tasks in the next MPT. In condition component, one of the possible next Grid tasks is selected based on a condition. The change of execution path makes traditional static scheduling schemes not feasible. However, MPT is constituted during run-time. It can support condition component well by adding condition constraint. Loop task will be iterated for many times in a Grid Workflow. There are two kinds of loop tasks- fixed loop and condition loop. The iteration times of fixed loop task is predefined and will not change in runtime. Thus we can stretch a fixed loop task to a sequence of tasks which is fully supported by MPT. The condition loop task's execution times depend on a condition expression. By adding the execution condition constraint, MPT can support condition loop task well.

4.2 Scheduling Algorithm

The scheduling action is executed in case of the following occurrences of events: 1) a new workflow request; 2) the completion of one task; 3) the failure of one task execution or the violation of its QoS tol-

erance bound. The set of tasks being scheduled in one scheduling action is the set of "maximum parallel tasks" at the scheduling time.

The scheduling algorithm has four steps. First step is to find the set of current "maximum parallel tasks"– T_m. Next is to calculate the execution price $p_{i,j} = \min_i P(i,j)$ for all tasks $t_j \in T_m$. Third is to sort $p_{i,j}$ increasingly and place them in a queue Q_m. Last is to schedule the queue Q_m one by one until the capacity of resource is reached or queue is empty.

Algorithm 2 Simple minimum scheduling algorithm.

```
find the set of "maximum parallel tasks"– T_m
n ← number of tasks in T_m
p_{i,j} ← min_i P(i,j), ∀i, t_j ∈ T_m
sort p_{i,j} in queue Q_m in increasing order
for k = 1; k ≤ n; k++ do
  take i, j from Q_m[k]
  if c_i > r_j and the bid is still valid then
    reserve required resources for t_j at resource i
  else
    break;
  end if
end for
```

Algorithm 2 is quite fast. However, it cannot guarantee the scheduled result is the optimal one with minimum total cost. Other possible scheduling algorithms include genetic algorithm (GA) and simulated annealing (SA). The common shortcoming of GA and SA is their high computation complexity. Here we propose a "minimum-penalty" iterative algorithm. As shown in Algorithm 2, $p_{i,j}$ is the minimum value of all $P(i,j)$. We define "penalty" as in following equation,

$$Pen(k,j) = P(k,j) - p_{i,j}, k \neq i \quad (4)$$

The pseudo code of minimum-punish algorithm is shown as Algorithm 3.

Algorithm 3 Minimum punish algorithm.

```
1:  n ← number of tasks in T_m
2:  for j = 1 to n do
3:    find p_{i,j}
4:  end for
5:  check the capacity validness of schedule {p_{i,j}}
6:  if schedule {p_{i,j}} is valid then
7:    return optimal schedule {p_{i,j}}
8:  else
9:    repeat
10:     find resource i_m whose capacity is the mostly
        violated
11:     find the set of tasks(S(i_m)) assigned to re-
        source i_m
12:     from S(i_m), find the new schedule (i_n, j_m)
        with minimum penalty and without new ca-
        pacity violation
13:     replace schedule (i_m, j_m) with (i_n, j_m)
14:   until all resource capacity constraints are sat-
      isfied
15: end if
```

5 NUMERICAL RESULTS

5.1 Simulation Scenario

A series of simulation case studies have been performed to evaluate the effectiveness of our new Grid scheduling algorithm. In the first case, we use a small workflow 1, which consists of ten abstract Grid tasks. The abstract task model contains the computation and QoS requirements of the task; while the resource model has following parameters: computation capability (CPU cycle), capacity, cost and QoS level provided by the resource. The computation capabilities and costs are randomly changed with time.

The evaluation metrics used in our simulation include makespan (workflow finish time) and cost. We compare these two metrics exhibited by our MPT scheduling algorithm with that of static and simple level decomposition algorithms(Deelman et al., 2004).

Besides the simple example workflow 1, we have also simulated a more complicated workflow consisting of 128 tasks as workflow example 2. Workflow example 2 consists of three loop tasks, 21 split tasks and 18 merge tasks. Figure 3 shows the makespan result of all three algorithms on two example workflows. Obviously our scheduler has the minimum makespan, while one-by-one scheduler has the greatest makespan. As a static approach, one-by-one scheduler performs the worst in a dynamic grid scenario. The tasks of next level should start after all tasks of current level have been completed. If there is a task A requiring much longer execution time than that of other tasks in current level, there will be a long period with only one task running since the next level tasks cannot start until task A finishes. As shown in Figure 4, our scheduler achieves the lowest cost for both example workflows; while level-decomposition algorithm experiences the highest cost. The result is not surprising since only our scheduler has considered the cost in scheduling. Furthermore, the longer the time of their tasks occupying resources, the more the users should pay.

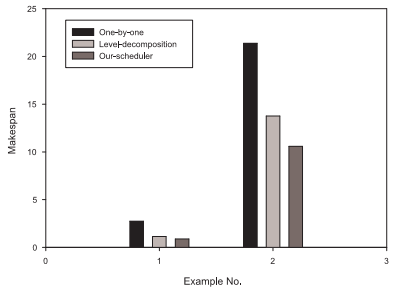

Figure 3: Makespan of example flows.

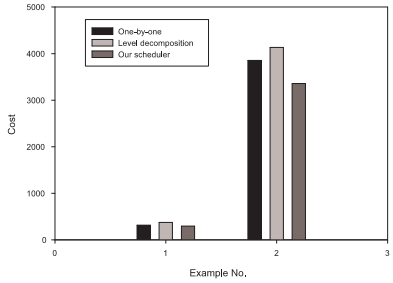

Figure 4: Cost of example flows.

6 CONCLUSION

In this paper, we propose a "temporal decomposition" scheme to decouple the whole large workflow scheduling problem to sub-workflow scheduling problem. An added advantage of our scheme is its adaptability to dynamic grid resources. The sub-workflow schedule is chosen according to the latest states of grid resources, instead of the states at the start time of workflow. We have also developed an scheduling algorithm to solve sub-workflow scheduling problem with resource constraints. The preliminary numerical results demonstrate that our scheme outperforms "one-by-one" and simple "level decomposition" schemes in both makespan and system cost. The performance of our scheduler in the presence of grid resource failures is interesting and will be investigated in our further work.

REFERENCES

Cardoso, J., Miller, J., Sheth, A., and Arnold, J. (2004). Modeling quality of service for workflows and web service processes. *Web Semantics Journal: Science, Services and Agents on the World Wide Web*, 1(3):281–308.

Cheng, J. Q. and Wellman, M. P. (1998). The WALRAS algorithm: A convergent distributed implementation of general equilibrium outcomes. *Computational Economics*, 12:1–24.

Deelman, E., Blythe, J., Gil, Y., and Carl Kesselman, e. (2004). Pegasus: Mapping scientific workflows onto the grid. In *AxGrids 2004, LNCS*, volume 3165, pages 11–20. Springer Berlin / Heidelberg.

Smale, S. (1976). Convergent process of price adjustment and global newton methods. *American Economic Review*, 66(2):284–294.

Wolski, R., Plank, J. S., Brevik, J., and Bryan., T. (2001). Gcommerce: Market formulations controlling resource allocation on the computational grid. In *IPDPS '01: Proceedings of the 15th International Parallel & Distributed Processing Symposium*, page 46, Washington DC. IEEE Computer Society.

Yu, J. and Buyya, R. (2005). A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3-4):171–200.

Yu, J., Buyya, R., and Tham, C. K. (2005). QoS-based Scheduling of Workflow Applications on Service Grids. In *Proc. of the 1st IEEE International Conference on e-Science and Grid Computing*, Melbourne, Australia.

Zhang, S., Gu, N., and Li., S. (2004). Grid workflow based on dynamic modeling and scheduling. In *Information Technology: Coding and Computing, 2004. Proceedings of 2004 International Conference on*, volume 2, pages 35–39.