

Scheduling Multiple Parameter Sweep Workflow Instances on the Grid

Sucha Smachat, Maria Indrawan, Sea Ling, Colin Enticott and David Abramson

Faculty of Information Technology

Monash University

Melbourne, Australia

*{sucha.smachat, maria.indrawan, chris.ling, colin.enticott,
david.abramson}@infotech.monash.edu.au*

Abstract

Due to its ability to provide high-performance computing environment, grid has become an important infrastructure to support eScience. To utilise the grid for parameter sweep experiments, workflow technology combined with tools such as Nimrod/K are used to orchestrate and automate scientific services provided on the grid. As parameter sweep workflow needs to be executed numerous times, it is more efficient to execute multiple instances of parameter sweep workflow in parallel. However, this parallel execution can be delayed as every workflow instance requires the same set of resources leading to resource competition problem. Although many algorithms exist for scheduling grid workflow, there is little effort in considering multiple workflow instances and resource competition in the scheduling process. In this paper, we proposed a scheduling algorithm for parameter sweep workflow based on resource competition. The proposed algorithm aims to support multiple workflow instances and avoid allocating resources with high resource competition to minimise delay due to the blocking of tasks. The result is evaluated using simulation to compare with the existing scheduling algorithm.

1. Introduction

Grid has become an important infrastructure to support eScience due to its ability to provide high-performance computing environment. To orchestrate and automate scientific services provided in the grid, workflow technology has been introduced into the grid domain. Scientific processes can be captured in scientific workflows which are then executed by scientific workflow management systems [1-3].

In order to execute scientific workflow in grid, the tasks in a scientific workflow need to be allocated to

grid resources. Therefore, the scheduling of workflow tasks onto grid has become an important issue and has been researched continuously. Consequently, there are many existing grid workflow scheduling algorithms proposed for different workflow applications which utilise various heuristics and techniques.

However, most existing grid workflow scheduling algorithms [4-6] do not take into account the dependency of tasks to specialised resources, nor do they consider popularity of a resource. For example, allocating a task (that takes a long time to complete) to a resource that is shared by many tasks might delay the overall execution. The limitation is amplified in the case of parallel execution of parameter sweep workflow such as in Quantum Chemical Calculations experiment [8]. This is due to the fact that in a parameter sweep workflow every instance of workflow comes from the same workflow definition. Thus these instances require the same type of resources. In other word, they may compete for the same set of resources. The situation may even be more serious if there is a competition for a scarce specialist resource.

The second concern of the existing scheduling algorithms is their limited support for scheduling multiple workflow instances [7]. For parameter sweep execution this poses serious performance issue. Without this support, in the worst case, the parallel execution of the parameter sweep workflow instances will be reduced to sequential execution.

Therefore, placing a close attention to resource requirement of a task and resource allocation during the design of the scheduling algorithm, in our opinion, will lead to the development of an efficient scheduling algorithm for parameter sweep workflow.

In this paper, we proposed a grid workflow scheduling algorithm based on resource competition among the tasks in the workflow. The proposed algorithm aims to support scheduling for multiple workflow instances of the same workflow definition

and to minimise overall workflow execution time by avoiding resource allocation that will block or delay the execution of other tasks based on the resource requirement.

The rest of this paper is organised as follows. Section 2 describes the related work including grid workflow, Nimrod and Kepler Systems, and existing scheduling algorithms. Section 3 describes our scheduling objective and the definitions in our algorithm which is described in Section 4. Section 5 presents the simulation result while Section 6 concludes this paper with the direction of future work.

2. Background

2.1 Grid and workflow

Workflow technology has been introduced into scientific domains to help scientists perform their work in the grid environment [2, 3]. These scientific workflows capture scientific processes and are executed automatically by scientific workflow management system such as Kepler [1] and Pegasus [8] which are designed specifically for scientific purposes.

Inherently, scientific workflows usually require high computation power [9, 10]. Grid environment offers high computation power through the coordination of distributed resources. Scientific workflow management systems that are grid-aware can use the grid services to distribute tasks in scientific workflows to be executed by available resources in the grid [10]. In order to distribute tasks to resources, a scientific workflow needs to be mapped to a grid workflow. Workflow scheduling is required so that the execution of scientific workflow can meet the constraints or criteria such as execution time and cost [7].

Among several scientific workflow applications on the grid, there is a type of scientific experiment which involves repeating the same process to search for optimal parameter set. This process is called parameter sweep and can be automated using tool such as Nimrod/K [11] which is implemented on the Kepler system [1].

2.2 Nimrod and Kepler systems

Kepler [1] is an open source, scientific workflow application developed by a cross-project collaboration led by UC Davis, UC Santa Barbara and UC San Diego. Kepler focuses on providing a common environment for eScientists to share, link and execute different models. The modular environment encourages collaboration by allowing parties to

contribute modules to a much larger workflow which in turn increases the value of the project as a whole.

The Nimrod toolset [12] is a project from Monash University that focuses on providing distributed computing and parametric tools to eScience applications. Like Kepler, Nimrod has been used to run applications from a wide range of fields recently including geophysics [13] and biochemistry [14], and has performed very large scale parametric studies across a large cross-grid experiment in crystallography [15]. The parametric tools offered by Nimrod include parameter sensitivity software that measures how each parameter is affecting the outcome. Another one of the tools can search parameter sets to optimise an application's result. Parametric studies execute the same application many times, varying the input parameters and producing many jobs. The Nimrod toolset also provides a tool called Nimrod/G which executes jobs using different Grid middleware like Globus, Condor and SGE. Within one experiment, Nimrod can access multiple resources, and balance the pool of jobs across them. The scheduler in Nimrod/G gives the user the option to run all the jobs as quickly as possible or with time and cost constraints and the scheduler will endeavour to do the best it can. This kind of scheduling becomes more complicated with workflows, as a task can be made up of multiple smaller grid tasks that share grid resources.

Nimrod/K is a project that combines the features of the Nimrod toolset with the Kepler environment by introducing a grid workflow framework. As mentioned earlier, Kepler provides tools for a workflow developer to access grid resources within a workflow. The focus of the Nimrod/K framework is to go one step further and move those services away from the workflow creator and put them into the framework. This creates a more institutive environment by focusing the workflow creator's attention on connecting the components together and not on what middleware is needed to make them work. For example, if a component on a workflow creates an output file on a grid resource, that file should be copied to the next resource used by the next component on the workflow. A grid scheduler can decide where to move the file if the next component can be scheduled on a selection of Grid resources.

2.3 Grid workflow scheduling algorithms

To execute a grid workflow, an instance of a grid workflow is created and then the task instances within it are scheduled to the resources available in the grid for execution. There exist many techniques for the scheduling of grid workflow with different scheduling objectives such as performance, cost, and resource utilization. We focus on the performance objective of

minimising the execution time or makespan. That is, the time taken to execute the workflow from the beginning of the first task to the completion of the last task. This objective is common in the grid workflow scheduling domain.

The three well-known scheduling algorithms, Min-Min, Max-Min, and Sufferage, were proposed by Maheswaran et al. [6]. These three algorithms were designed as batch mode scheduling. They first create a list of tasks that are ready to be executed; this process is called “task prioritising” phase [5]. In the second phase, the tasks in the list are scheduled to resources based on a heuristic; this process is called the “resource selection” phase [5].

The task prioritising phase in the Min-Min algorithm involves the creation of a list of tasks that are ready to be executed. The algorithm then computes the Estimated Completion Time (ECT) of each task for each suitable resource. A resource that can execute a task with Minimum ECT (MCT) is then chosen as the most suited resource for that task. These task and resource are then paired. In the resource selection phase, the resource-task pair with lowest value of MCT is scheduled first. The process is repeated until every task is scheduled [5, 6]. The Max-Min algorithm is similar to the Min-Min algorithm except that it schedules a resource-task pair with the highest value of MCT first. The Sufferage algorithm uses the Sufferage value which is the difference in value between the lowest and second lowest MCT. The pair of task and resource with maximum Sufferage value is scheduled first [6]. The execution of the algorithm in practice gives priority to a task that would suffer the most if it is not executed first [16].

As an extension to Sufferage, Casanova et al. [16] proposed XSufferage algorithm for parameter sweep applications. Sufferage value in this algorithm is computed taking into account the time required to transfer data file. He et al. [17] extended Min-Min algorithm to take Quality of Service (QoS), in term of network bandwidth, into consideration. Etminani and Naghibzadeh [18] improved the Min-Min and Max-Min algorithms by using selective approach to utilise the advantage of both algorithms.

Liu et al. [19] proposed the Min-Min Average algorithm which uses the minimum average execution time for scheduling decision instead of the MCT. This algorithm also supports multiple workflow instances. However, they do not consider resource competition caused by numerous tasks in all workflow instances.

Apart from the batch mode scheduling algorithms previously discussed, Topcuoglu et al. [4] proposed the “Heterogeneous-Earliest-Finish-Time” (HEFT) to schedule workflow onto heterogeneous resources.

HEFT is a dependency mode scheduling which ranks the tasks in the task prioritising phase based on the average execution time of the tasks, the average time required for the transferring of data between the resources, and the position of the tasks in the workflow [5]. The tasks that are closer to the beginning of the workflow are ranked higher and are then scheduled first to the resource that can complete them earliest.

As an extension to HEFT, Sakellariou and Zhao [20] proposed an improved hybrid version of HEFT by grouping the independent tasks together after ranking and sorting them. Then the tasks in each group can be scheduled using “Balanced Minimum Completion Time” (BMCT) algorithm which can move tasks between resources to balance resource load.

Shi and Dongarra [21] proposed the “Scheduling algorithm for heterogeneous processors with Different Capabilities” (SDC) to deal with the situation where a task can only be executed by certain resources. Tasks with “scarce capable resources” - the tasks that fewer resources are able to execute - are given higher priority so that they are scheduled and will obtain the required resources first before other tasks that can be executed by many other resources [21]. However, since the ranking in HEFT based algorithms is based on the dependency of tasks [5], the tasks with scarce capable resources further down the workflow may still be blocked [4, 21]. For example, at the beginning of a workflow, a scarce resource R might be allocated to a very long task T . This can block the tasks further down the workflow which are independent of T but dependent on R . In addition, the definition of scarce resource in this work (“scarce capable resource” mentioned above) does not include the resources which are needed by several tasks. This necessitates the need to include the notion of resource competition into scheduling algorithm.

To the best of our knowledge, there is currently no grid workflow scheduling algorithm that both supports multiple instances and considers resource competition. To efficiently schedule parameter sweep workflow for parallel execution, it is necessary to consider resource competition issue and at the same time support multiple workflow instances. In this paper, we adopt the notion of “resource scarcity” defined in [21] and propose a novel multiple grid workflow instances scheduling algorithm based on resource competition.

3. Objective and definitions

In this section we state our objective and define the terms and notations for the proposed scheduling algorithm.

3.1 Scheduling objective

Given a workflow represented as DAG (Directed Acyclic Graph), the number of instances to be executed, and a set of resources, our objective is to *find a schedule that allocates capable resources to tasks in multiple instances of a grid workflow based on resource scarcity and resource competition in a way that minimises the makespan of the entire execution of all workflow instances.*

3.2 Grid workflow

A workflow is a collection of tasks which are connected together to achieve a particular goal. We model grid workflow as Directed Acyclic Graph (DAG). At this stage, loop structure is not taken into consideration. We extend grid workflow with the notion of capable resource and define grid workflow $W = (V, E, R)$ where V is a set of nodes representing tasks in the workflow; E is a set of edges representing the precedence dependencies between tasks or nodes in the workflow; and R is a set of resources for executing tasks in the workflow.

Before the grid workflow is executed, the instance of that particular workflow definition is first created. Since we are dealing with multiple workflow instances, it is important to note that a task instance in one workflow instance is different from another task instance in another workflow instance even though they are of the same task in the workflow definition.

3.3 Parameter sweep workflow

As mentioned earlier, a parameter sweep workflow is a workflow that is executed several times to find the optimal set of parameters. A parameter sweep workflow instance is created and is assigned a unique parameter combination for the execution. These parameter sweep workflow instances are semantically independent of each other.

We extend the grid workflow definition in section B and define parameter sweep workflow as $PS = (V, E, R, P)$ where P is a set of parameter combinations which can be used to determine the total number of workflow instances to be executed.

3.4 Grid resource

In order to execute tasks in the workflow, each task must be assigned to a grid resource. Similar to [21], we assume that each task instance can only be executed by certain resources. During scheduling, a set of current resources that can execute task instance t_i or *Capable Resources* for t_i is denoted as $CR(t_i) \subseteq R$.

In grid environment, a resource may refer to a cluster resource which has multiple execution nodes. It is therefore necessary to consider the number of available nodes on each resource during scheduling. We define a function to retrieve the number of available nodes in a resource r as

$$nodes(r) = \text{number of available nodes in } r \quad (1)$$

where $r \in R$. In addition, we assume that each node can only execute one task at a time. There are three performance metrics associated with each resource which are used in this paper [5]. These metrics are listed in table 1. In the actual grid environment, the estimated execution time can be obtained from Nimrod execution history and the estimated wait time can be obtained by determining the task queue in each resource.

Table 1. Terms and definitions

Term	Definition
EET	Estimated Execution Time: The time taken for a resource to execute a task
EWT	Estimated Wait Time: The time until a resource can start executing a task
ECT	Estimated Completion Time: The time taken for a resource to complete a task after the task is allocated to it.

3.5 Resource competition and resource scarcity

We adopt the notion of “Resource Scarcity” from [21]. Note that we modify the term “Capable Processor” to “Capable Resource” to make the algorithm more generic. As a grid resource can govern multiple execution nodes, we denote the number of capable resource nodes for task instance t_i as $CRN(t_i)$ which is the sum of the number of nodes in each capable resource in $CR(t_i)$.

$$CRN(t_i) = \sum_{k=1}^{|CR(t_i)|} nodes(r_k) \quad (2)$$

where $r_k \in CR(t_i)$. We then apply the calculation of resource scarcity from [21]. The resource scarcity (RS) for each task instance t_i ($i \in N$) is defined as

$$RS(t_i) = \frac{CRN(t_i)}{\sum_{j=1}^{|R|} nodes(r_j)} \quad (3)$$

where $r_j \in R$. Resource scarcity is the ratio between the number of capable resource nodes for t_i and the number of all resource nodes. The lower the resource scarcity value means there are fewer resource nodes that can execute the task. This value is used to prioritise tasks in our algorithm.

We also prioritise resources based on resource competition so we can avoid allocating the resources that have high degree of competition i.e. the resources that are required by many tasks. The resource competition (RC) of resource r_j is defined as

$$RC(r_j) = \frac{\sum_{i=1}^{\|RT(r_j)\|} \frac{1}{CRN(t_i)}}{t_{total}} \quad (4)$$

where $t_i \in RT(r_j)$ and t_{total} is the total number of task instances that are to be scheduled. $RT(r_j) \subseteq T$ is the set of requiring unscheduled task instances that can be executed by r_j ($j \in N$). Resource competition is the ratio between the summation of the inverse ratio of the number of capable resource nodes for each requiring unscheduled task instance and the number of all unscheduled task instances across all workflow instances. The higher the resource competition value means there are more task instances on average that require that particular resource for execution.

3.6 Ranking function

In order to select a resource for a task, we use a ranking function based on the resource competition value and the Estimated Completion Time (ECT) of each resource. At this stage we exclude communication time or the time taken for file transfer between resources from our consideration. The ECT of a task instance t_i when executed by a resource r_j ($i, j \in N$) is calculated using the following formula [5, 6].

$$ECT(r_j, t_i) = EET(r_j, t_i) + EWT(r_j) \quad (5)$$

Resource competition value alone cannot be used as the rank as there might be a resource with low competition value that takes much longer to execute a task. Therefore, the ranking function is defined as the product of resource competition value and ECT . The rank of each resource r_j executing a task instance t_i ($i, j \in N$) is calculated using the following formula.

$$Rank(r_j) = RC(r_j) \bullet ECT(r_j, t_i) \quad (6)$$

Note that a lower rank value is better since it means the resource has lower competition and can also execute the task faster. With this ranking function, a resource with a much lower ECT and a slightly higher resource competition value can be selected.

4. Parallel algorithms for parameter sweep

To efficiently schedule parallel parameter sweep workflow, a scheduling algorithm must be able to support multiple workflow instances. Most of the current batch scheduling algorithms can be adopted to handle multiple workflow instances for parameter

sweep easily. This section describes how an existing grid scheduling algorithm can be adapted to handle multiple workflow instances. We choose the Min-Min algorithm [6] to illustrate this adaptation due to its popularity and good performance [17, 18].

In addition, a new scheduling algorithm based on multiple performance metrics is presented. Currently, resource competition and the estimated completion time are the two metrics used to illustrate the proposed scheduling algorithm. In the future, additional metrics such as file transfer time will be considered. We start the discussion by presenting the multiple performance metrics algorithm.

4.1 Scheduling algorithm based on multiple performance metrics.

To support multiple workflow instances, we design our algorithm as a batch mode scheduling so that we can gather and schedule ready tasks from multiple workflow instances. We separate our algorithm into three phases namely *Instance Generation Phase*, *Task and Resource Prioritising Phase* and *Resource Selection Phase* as shown in Fig. 1.

In the Instance Generation Phase, as parameter sweep workflow can involved numerous workflow instances, a control is required so that there are not too many instances of parameter sweep workflow in comparison to the number of available resources at any given time. We currently compare the total number of unscheduled task instances across all the created instances against the total number of free resource nodes. If the number of unscheduled task instances is equal to or less than the number of free resource nodes, a new workflow instance is created and is assigned the next parameter combination (lines 2-5).

After the workflow instances are generated and the resources are updated, a list of capable resources $CR(t_i)$ for each task instance is created (line 8). The resource scarcity of each unscheduled task is calculated using (3) and the resource competition of each resource is calculated using (4). After that, the task instances whose predecessor task instances have completed execution are determined and added to the batch RB (line 12). Once all the ready task instances are obtained, the task instances in the ready batch RB are sorted in non-decreasing order based on RS . If there are two or more task instances with the same RS value, the tie is broken by the average ECT in non-decreasing order.

In the Resource Selection Phase, the first task instance t_l in the batch is removed from the batch to be scheduled. This task is the task with the lowest RS in the batch i.e. the task that lowest number of resources can execute. The ranking function (6) is applied to

Phase	Input: Parameter sweep workflow, total number of instances, resources Output: Schedule for executing multiple instances of parameter sweep workflow
Start	1 while there are more instances to schedule or set of unscheduled tasks T is not empty do
Instance Generation	2 while number of unscheduled tasks in set $T \leq$ number of free resource nodes 3 Generate one workflow instance 4 Add task instances from the new workflow instance to set T 5 end while
Task & Resource Prioritising	6 Update available resources in set R 7 for each task instance t_i ($i \in N$) in set T do 8 Find capable resources for t_i and add them to set $CR(t_i)$ 9 Calculate $RS(t_i)$ using (3) 10 end for 11 for each resource r_j ($j \in N$) in set R do calculate $RC(r_j)$ using (4) 12 Determine the task instances in set T that are ready to execute and add them to set RB 13 Sort tasks in set RB in non-decreasing order based on RS
Resource Selection	14 while set of ready task instances RB is not empty do 15 Get the first task instance t_i from RB 16 for each capable resource r_j in $CR(t_i)$ do 17 Calculate ECT using (5) 18 Calculate rank using (6) 19 end for 20 Assign t_i to the resource r_{min} with the lowest rank 21 Update EWT of r_{min} 22 Remove t_i from RB 23 end while
Exit	24 Wait for next scheduling round 25 end while

Figure 1. The proposed algorithm

calculate the ranks of all capable resources of t_i (line 16-19). Finally t_i will be allocated to the resource with the lowest rank and then the EWT of the allocated resource is updated. This process repeats until the batch is empty. After all the task instances in the batch are scheduled, the scheduling process stops and waits until the next scheduling round. The next scheduling round is triggered by one of the following conditions: a certain period of time elapses; a new resource joins the grid which might consequently trigger the generation of a new workflow instance; or a task instance finishes execution which might enable its succeeding task instances.

Note that the resource competition is calculated based on the unscheduled task instances in the entire workflow. Therefore, the ranking function is affected by the unscheduled tasks further down the workflow so the algorithm can avoid allocating the resource with expected high competition. In addition, a much faster resource with a slightly higher RC value can be allocated first as it may become available again before the tasks down the workflow are ready to execute.

4.2 Parallel Min-Min algorithm

We apply the Min-Min algorithm to support parallel multiple instances for comparison. As both algorithms are batch scheduling, the *Task Prioritising Phase* and the *Resource Selection Phase* in Fig. 1 can be replaced by the Min-Min algorithm easily. However, the same *Instance Generation Phase* and the same triggers for advancing to the next scheduling round are used in

order to adapt the Min-Min algorithm to support multiple workflow instances.

5. Simulation result

This section presents the simulation result of the proposed algorithm implemented as a prototype scheduler in Nimrod/K. We present the result in comparison to the original sequential Min-Min algorithm and the applied Parallel Min-Min algorithm.

5.1 Nimrod/K

The Nimrod/K framework provides workflow schedulers the access to the underlying features of Grid middle tools as well as Nimrod's parameter exploration tools. It gives a scheduler access to all the file transfers, compute resource scheduling and file streaming that need to be scheduled. Earlier work from the Nimrod/K [11] project included a system of having threads included in a Kepler workflow. This included threading the whole workflow or partial section of the workflow. Nimrod/K also has the APIs for a scheduler to support multiple Grid middleware.

In addition to this framework, it also allows the creation of a simulated grid resource. A Grid simulation actor can report the simulated grid resources as if they were real resources. When fired, it can simulate the time required to perform the action. Just by swapping the actors in the workflow it can perform a simulation

or run a real experiment. A scheduler only has to be developed once.

This framework allows exploration of scheduling ideas using many different Grid APIs and using real world applications.

5.2 Simulation environment

Nimrod/K is used as the simulation environment with the scenario in Fig. 2. The *Parameter Sweep Actor* is used to generate workflow instances during run-time. The tasks in the scenario are simulated by dummy actors. The dummy actor simulates task execution by going into sleep state for a period of time equal to the estimated completion time (*ECT*) as if it is executed by a particular resource. Resource simulator is created so that it is possible to control the resources in the simulation. Each resource is set to have only one execution node so that the *EWT* of each resource can be approximated.

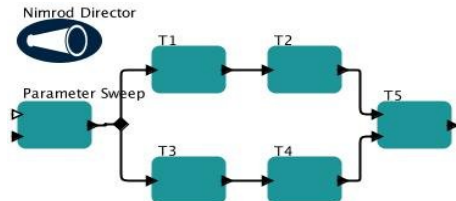


Figure 2. Simulation scenario

To demonstrate the proposed algorithm, we design two simulation settings. Simulation A assumes that every task can be executed by any resources while Simulation B assumes that each task can be executed by certain capable resources. Table 2 and table 3 list the estimated execution times (in second) of each resource for the tasks in the two simulation settings. The simulation is set to execute 1, 5, 10, 20, and 50 parameter sweep workflow instances.

Table 2. EETs for Simulation A

	t_1	t_2	t_3	t_4	t_5
r_1	8	7	6	9	5
r_2	9	6	8	10	7
r_3	8	5	4	6	4
r_4	7	6	6	8	6
r_5	7	4	5	7	4

Table 3. EETs for Simulation B

	t_1	t_2	t_3	t_4	t_5
r_1	8	-	6	-	5
r_2	9	6	-	-	-
r_3	8	5	4	-	-
r_4	7	-	-	8	6
r_5	-	-	5	-	4

5.3 Simulation result and discussion

Table 4 presents the average improvement of makespan that Parallel Min-Min and our proposed algorithm made in comparison to the sequential Min-Min. The performance is reported for both simulations.

Table 4. Average improvement over sequential Min-Min

Simulation	Parallel Min-Min	Proposed Algorithm	Improvement against Parallel Min-Min
A	36.38%	36.40%	0.06%
B	44.81%	50.40%	12.48%

The results show that both algorithms that considered parallelism outperform the sequential Min-Min as expected. The improvement is up to 50.40% when an execution plan that involves specialist resource is handled by our proposed algorithm. The proposed algorithm shows similar performance to Parallel Min-Min in simulation A due to the fact that there is no specialist resource and the resource competition for each resource is similar. Hence in this situation we expect that the proposed algorithm will perform similar to Parallel Min-Min. The value of *Rank* in (6) will have similar *RC*, hence rank is mainly determined by *ECT*. In simulation B, in contrast, the proposed algorithm performs better than Parallel Min-Min because of the existence of specialist resources such as r_4 as it is the only resource capable of executing task t_4 . In this type of situation the *RC* values will vary affecting the overall ranking process of the task allocation.

6. Conclusion and future work

In this paper, we propose a grid workflow scheduling algorithm based on resource competition for parallel parameter sweep workflow. Combined with the features of Nimrod/K [11], the proposed algorithm can control and support dynamic scheduling of multiple workflow instances. The proposed algorithm can avoid allocating resources with high competition to reduce the delay due to blocking of the other tasks across multiple workflow instances. However, the proposed algorithm especially the ranking function needs to be refined. In addition, further improvement can be made using resource exclusion as suggested in [22] and utilising the structure of the workflow to refine the resource competition calculation.

We plan to take the time for transferring files between resources (communication time) into consideration. Also, we aim to extend the proposed algorithm to support loop and conditional structures in grid workflow. The algorithm is intended to be further

tested using real parameter sweep workflows in actual grid environment.

References

- [1] B. Ludäscher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger, M. Jones, E. A. Lee, J. Tao, and Y. Zhao, "Scientific workflow management and the Kepler system," *Concurr. Comput. : Pract. Exper.*, vol. 18, pp. 1039-1065, 2006.
- [2] J. Zhang, D. Pennington, and W. Michener, "Using Web Services and Scientific Workflow for Species Distribution Prediction Modeling," in *Advances in Web-Age Information Management*, 2005, pp. 610-617.
- [3] Z. Jianting, "Ontology-Driven Composition and Validation of Scientific Grid Workflows in Kepler: a Case Study of Hyperspectral Image Processing," in *Proceedings of the 5th International Conference on Grid and Cooperative Computing Workshops (GCCW '06)*, 2006, pp. 282-289.
- [4] H. Topcuoglu, S. Hariri, and W. Min-You, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, pp. 260-274, 2002.
- [5] J. Yu, R. Buyya, and K. Ramamohanarao, "Workflow Scheduling Algorithms for Grid Computing," in *Metaheuristics for Scheduling in Distributed Computing Environments*, 2008, pp. 173-214.
- [6] M. Maheswaran, S. Ali, H. J. Siegel, D. Hensgen, and R. F. Freund, "Dynamic matching and scheduling of a class of independent tasks onto heterogeneous computing systems," in *Proceedings of the 8th Heterogeneous Computing Workshop (HCW '99)*, 1999, pp. 30-44.
- [7] M. Wiczorek, A. Hoheisel, and R. Prodan, "Taxonomies of the Multi-Criteria Grid Workflow Scheduling Problem," in *Grid Middleware and Services*, 2008, pp. 237-264.
- [8] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. S. Katz, "Pegasus: A framework for mapping complex scientific workflows onto distributed systems," *Sci. Program.*, vol. 13, pp. 219-237, 2005.
- [9] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers, "Examining the Challenges of Scientific Workflows," *Computer*, vol. 40, pp. 24-32, 2007.
- [10] J. Yu and R. Buyya, "A taxonomy of scientific workflow systems for grid computing," *SIGMOD Rec.*, vol. 34, pp. 44-49, 2005.
- [11] D. Abramson, C. Enticott, and I. Altintas, "Nimrod/K: towards massively parallel dynamic grid workflows," in *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*. Austin, Texas: IEEE Press, 2008.
- [12] *Nimrod Toolkit*, <http://messagelab.monash.edu.au/Nimrod>, Accessed July, 2009.
- [13] S. Ayyub, D. Abramson, C. Enticott, S. Garic, and J. Tan, "Executing Large Parameter Sweep Applications on a Multi-VO Testbed," in *Proceedings of the 7th IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2007)*: IEEE Computer Society, 2007, pp. 73-80.
- [14] T. C. Peachey, N. T. Diamond, D. A. Abramson, W. Sudholt, A. Michailova, and S. Amirrazi, "Fractional factorial design for parameter sweep experiments using Nimrod/E," *Journal of Scientific Programming*, vol. 16, pp. 217-230, 2008.
- [15] B. Bethwaite, D. Abramson, and A. Buckle, "Grid Interoperability: An Experiment in Bridging Grid Islands," in *Proceedings of the 2008 4th IEEE International Conference on eScience*: IEEE Computer Society, 2008.
- [16] H. Casanova, A. Legrand, D. Zagorodnov, and F. Berman, "Heuristics for scheduling parameter sweep applications in grid environments," in *Proceedings of the 9th Heterogeneous Computing Workshop (HCW 2000)* 2000, pp. 349-363.
- [17] X. He, X. Sun, and G. v. Laszewski, "QoS guided min-min heuristic for grid task scheduling," *J. Comput. Sci. Technol.*, vol. 18, pp. 442-451, 2003.
- [18] K. Etmiani and M. Naghibzadeh, "A Min-Min Max-Min selective algorithm for grid task scheduling," in *Proceedings of the 3rd IEEE/IFIP International Conference in Central Asia on Internet (ICI 2007)*, 2007, pp. 1-7.
- [19] K. Liu, J. Chen, H. Jin, and Y. Yang, "A Min-Min Average Algorithm for Scheduling Transaction-Intensive Grid Workflows," in *Proceedings of the 7th Australasian Symposium on Grid Computing and e-Research (AusGrid 2009)*. Wellington, New Zealand: Australian Computer Society, Inc., 2009, pp. 41-48.
- [20] R. Sakellariou and H. Zhao, "A hybrid heuristic for DAG scheduling on heterogeneous systems," in *Proceedings of the 18th International Conference on Parallel and Distributed Processing Symposium*, 2004, pp. 111.
- [21] Z. Shi and J. J. Dongarra, "Scheduling workflow applications on processors with different capabilities," *Future Gener. Comput. Syst.*, vol. 22, pp. 665-675, 2006.
- [22] D. Kondo, A. A. Chien, and H. Casanova, "Scheduling Task Parallel Applications for Rapid Application Turnaround on Enterprise Desktop Grids," *Journal of Grid Computing*, vol. 5, pp. 379-405, 2007.