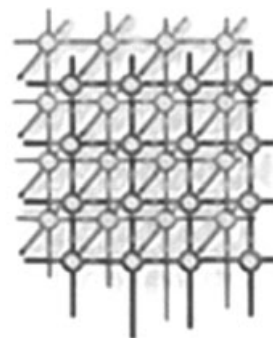


A throughput maximization strategy for scheduling transaction-intensive workflows on SwinDeW-G



Ke Liu^{1,2,*}, Jinjun Chen¹, Yun Yang¹ and Hai Jin²

¹*Centre for Information Technology Research, Faculty of Information and Communication Technologies, Swinburne University of Technology, P.O. Box 218, Hawthorn, Melbourne, Vic. 3122, Australia*

²*School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan 430074, China*

SUMMARY

With the rapid development of e-business, workflow systems now have to deal with transaction-intensive workflows whose main characteristic is the huge number of concurrent workflow instances. For such workflows, it is important to maximize the overall throughput to provide good quality of service. However, most of the existing scheduling algorithms are designed for scheduling of a single complex scientific workflow instance and are not efficient enough for scheduling transaction-intensive workflows. To address this problem, we propose a throughput maximization strategy (TMS), which contains two specific algorithms for scheduling transaction-intensive workflows at the instance and task levels, respectively. The first algorithm called Opposite Average Load tries to maximize the overall throughput by pursuing the overall load balance at the instance level, whereas the second algorithm called Extended Min–Min tries to further maximize the overall throughput at the task level by increasing the utilization rate of resources within each local autonomous group. The comparison and simulation performed on Swinburne Decentralized Workflow for Grid (SwinDeW-G), a peer-to-peer-based grid workflow environment, demonstrate that our strategy can improve the overall throughput significantly over existing scheduling algorithms when scheduling transaction-intensive workflows. Copyright © 2008 John Wiley & Sons, Ltd.

Received 3 February 2008; Accepted 6 February 2008

KEY WORDS: scheduling algorithms; transaction-intensive workflows; p2p-based grid workflow systems

*Correspondence to: Ke Liu, Centre for Information Technology Research, Faculty of Information and Communication Technologies, Swinburne University of Technology, P.O. Box 218, Hawthorn, Melbourne, Vic. 3122, Australia.

†E-mail: kliu@ict.swin.edu.au

Contract/grant sponsor: Australian Research Council; contract/grant numbers: DP0663841, LP0669660

Contract/grant sponsor: Swinburne Dean's Collaborative Grants Scheme 2007–2008

Contract/grant sponsor: Swinburne Research Development Scheme 2008



1. INTRODUCTION

The wide use of the Internet and the promotion of grid technology are changing our ways of thinking. The prosperity of e-business requires the ability to process transaction-intensive workflows. The most notable characteristic of these workflows is the large amount of concurrent relatively simple workflow instances and the fierce competition of resources. It is necessary to consider these new features when it comes to the design of scheduling algorithms.

However, most of the existing scheduling algorithms are designed for scheduling of a single complicated workflow instance rather than for multiple transaction-intensive workflows. To address this problem, we need to develop new solutions for scheduling of transaction-intensive e-business workflows.

The new algorithms must be able to support the scheduling of multiple instances of multiple workflows, because there are usually hundreds of thousands of instances running concurrently in an e-business environment. Therefore, the new algorithms should consider not only the completion time of each single instance, but most importantly the overall performance.

The overall performance can be defined in many aspects. Among them, we focus on the following aspects according to the characteristics of transaction-intensive workflows: (1) the overall load balance—it is obvious that an overall load-balanced system can provide better quality of service (QoS) for most users; (2) efficient resource utilization—the algorithms must manage the resource efficiently to deal with the fierce competitions of resources of the huge number of workflow instances; and (3) the overall throughput—the ultimate goal of transaction-intensive workflow scheduling is to maximize the overall throughput, even with a slight QoS degradation for some single instances.

To achieve the goals mentioned above, we propose a throughput maximization strategy (TMS), which contains two specific scheduling algorithms in different layers for transaction-intensive workflows. In detail, the Opposite Average Load algorithm (OAL) mainly deals with the instance scheduling and tries to maximize the overall throughput by pursuing the overall load balance. The Extended Min–Min algorithm (EMM) primarily deals with the task scheduling and tries to further maximize the throughput by increasing the utilization rate of resources within each local autonomous group. As we will illustrate later, these two algorithms can produce better scheduling results for transaction-intensive workflows than the original algorithms in both theory and practice.

The simulation performed later will demonstrate that this strategy has larger throughput over existing scheduling schemes when it comes to the scheduling of transaction-intensive workflows in a peer-to-peer (p2p) based grid workflow environment such as the Swinburne Decentralized Workflow for Grid (SwinDeW-G) [1].

The remainder of this paper is organized as follows. Section 2 describes the related work. Section 3 addresses the scheduling infrastructure. Section 4 discusses the details of scheduling algorithms. Section 5 shows the benefits of our work through comparison and simulation. Section 6 concludes our contributions and points out the directions for future work.

2. RELATED WORK AND PROBLEM STATEMENT

In this section, we describe the related work of workflow scheduling. Scheduling in distributed systems is NP-complete in general. There are many domain-specific algorithms aiming at finding good



though not optimal solutions for scheduling [2]. The typical grid workflow scheduling algorithms are overviewed in [3]. We will discuss the related work from different aspects and point out why new strategies are needed for transaction-intensive workflows.

In [4], many practical workflow systems are mentioned. As we take a look at the scheduling algorithms they use, it is possible to see that most of the practical algorithms are designed for scientific workflows and mainly deal with scheduling of a single complex workflow instance, such as the Min–Min Heuristic [5] and the Greedy Randomized Adaptive Search Procedure algorithm [6] used by Kepler [7], the Heterogeneous Earliest-Finish-Time algorithm [8] used by ASKALON [9], the Myopic algorithm [10] implemented by Condor DAGMan [11], a Hybrid algorithm [12] proposed by Sakellariou and Zhao and the Task duplication-based scheduling algorithm for Network of Heterogeneous systems [13] proposed by Bajaj and Agrawal. These algorithms are designed for scheduling complex scientific workflows and usually take a directed acyclic graph as their input. None of the algorithms are actually dedicated to transaction-intensive workflows and handle multiple instances of multiple workflows as required.

As a comparison, some scheduling algorithms use QoS-constrained strategies and seem to be more suitable for e-business workflows, such as the Back-Tracking algorithm [14] proposed by Menasc and Casalicchio, the Deadline Distribution algorithm [15] and Genetic algorithm [16] used by GridFlow [17] and the LOSS and GAIN algorithms used by CoreGrid [10]. However, most of these algorithms still care more about the scheduling performance of a single workflow instance, and the overall performances do not get enough attention.

According to Yu and Buyya [3], there are some algorithms that seek good scheduling results using other approaches, such as a scheduling algorithm considering background workload [18], a scheduling algorithm based on the prediction of historical record [19], a scheduling algorithm using prediction accuracy as a parameter during scheduling [20], a scheduling strategy based on the prediction of workload modelling [21] and a scheduling system called Grid Harvest [22] predicting the performance of available resources by the performance model parameters. Although these algorithms consider different aspects for their respective purposes, none of them are suitable for scheduling transaction-intensive workflows because the key features of transaction-intensive workflows, such as the overall load balance, efficient resource utilization and overall throughput, are not specifically taken into consideration.

In conclusion, although the algorithms mentioned above have their respective benefits in their particular application areas, none of them are particularly designed for transaction-intensive workflow scheduling and the key features of such workflows are not specifically taken into consideration. Thus, it is critical to develop scheduling algorithms dedicated to transaction-intensive e-business workflows that maximize the overall throughput as ultimate goal. This is the motivation behind the design for TMS for scheduling transaction-intensive workflows on the platform of SwinDeW-G.

3. SCHEDULING INFRASTRUCTURE OF TMS

In this section, we demonstrate the scheduling infrastructure of TMS. As Figure 1 shows, the system is divided into two layers. The scheduling groups are located in the lower layer. Each group consists of a super node and some ordinary nodes. The super node is in charge of the management of the group and tasks scheduling, whereas the ordinary nodes are responsible for the actual execution of

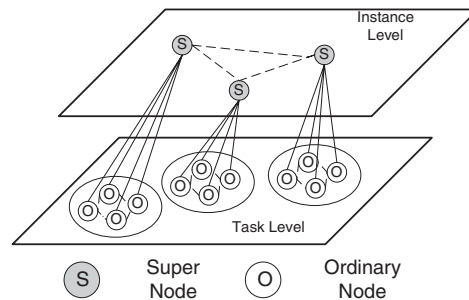


Figure 1. Scheduling infrastructure.

tasks. At the same time, the super nodes form the upper layer. The groups on the lower layer are usually formed according to the physical location. For example, the nodes in a cluster are more likely to form a group. Although the nodes in a group are managed by the super node in a centralized manner for the scheduling purpose, communication among the nodes is kept in a p2p manner to take advantage of the p2p technology.

The top portion of Figure 1 shows that all super nodes are organized into a mesh on the upper layer. On this level the workflow instances are scheduled by the cooperation of super nodes to achieve overall load balance. The relationship between the super nodes is loosely coupled due to the fact that the grid is usually dynamic with clusters and nodes joining or leaving at any time. Each super node maintains a neighbour list and exchanges necessary information with their neighbours periodically for maintenance and scheduling purposes.

Considering this system design and the characteristics of transaction-intensive workflows, we divide the scheduling process into two stages accordingly. On the first stage, considering the large amount of concurrent workflow instances, it would be better to distribute them evenly before scheduling them directly in the groups to which they are submitted to. In fact, before an instance is actually scheduled inside a group, the super node will decide whether to accept it or pass it to another super node according to applied algorithms. Generally speaking, the purpose of these algorithms is to achieve overall load balance. On the second stage, the accepted instance will be submitted to the selected group. Every group is an autonomous scheduling region with the super node controlling all the resources belonging to it. The centralized control grants the super node the ability to manage the resources efficiently so that it can deal with the fierce competitions for resources. Of course, the ultimate goal of scheduling at this level is to further maximize the overall throughput, even with a slight QoS degradation for some single instances.

4. THE OAL AND EMM ALGORITHMS

In this section, we detail the OAL and EMM scheduling algorithms. The former is designed to achieve overall load balance and the latter mainly maximizes the utilization rate of resources to increase the overall throughput. As mentioned before, these two algorithms are realized on the upper and the lower layers, respectively. We will discuss them in Sections 4.1 and 4.2, respectively.



4.1. OAL scheduling algorithm

Before we present the algorithm, we address a simple example. Figure 2(a) shows a mesh made of super nodes A–G. The value marked for each node represents the current load value. Figure 2(b) shows the distance value of each node from node G.

At this level, each super node uses a specific scheduling algorithm to achieve the overall load balance at the instance level, for example, the gossip-based scheduling algorithm. This algorithm is the most popular one used by many practical distributed systems and will be denoted as the original Gossip algorithm in the remainder of this paper.

A typical Gossip algorithm proceeds at two phases. For workflow scheduling, in the first phase, each super node chooses some random neighbours and exchanges load values with them in parallel. Then, at the second phase, each super node inserts its load value into the messages it has received and then propagates them in the next round. In this manner, every super node knows the load value of its neighbours. Thus, when an instance is submitted to the super node, it will accept the workflow instance if it is the least loaded super node of its surrounding area or pass the instance to its least loaded neighbour.

As the example continues, suppose that 47 instances are submitted to node G. After applying the original Gossip algorithm, the scheduling result is displayed in Figure 3. It can be clearly seen that this algorithm needs great improvement.

On the one hand, the original Gossip algorithm mentioned above is obviously designed for ordinary workflows scheduling. However, considering the large number of workflow instances of transaction-intensive workflows, we must change the one-by-one mode to batch mode. On the other

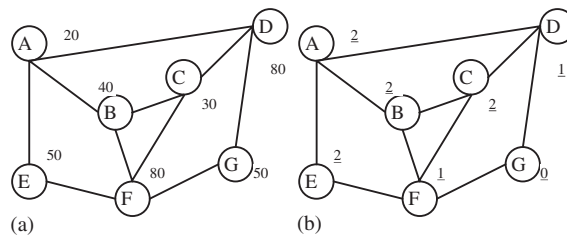


Figure 2. Simple mesh example.

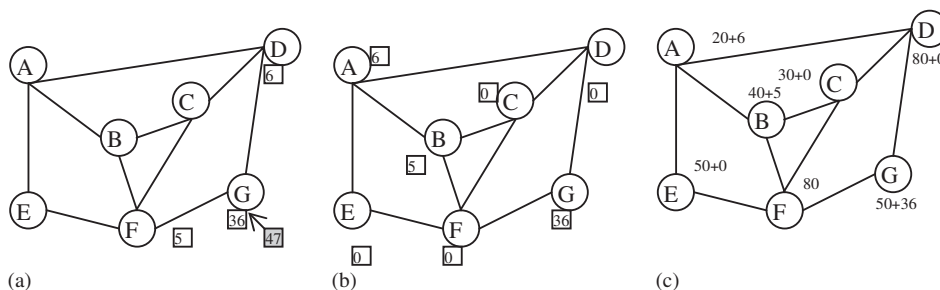


Figure 3. Scheduling result of original Gossip algorithm.



hand, we can use different parameters for a better scheduling result. Thus, we can improve the above algorithm as follows.

Each super node maintains the load value of the group it manages and the OAL value of its direct neighbours. For a specific node A, the OAL value of its neighbour B is the average load value of B and all its neighbours except A. The OAL can present the average load of the opposite direction, which can guide the scheduling procedure later. Table I summarizes the OAL values of every node in Figure 2. Of course, the super node exchanges load values with each neighbour for the calculation of OAL values periodically.

To prevent instances from being distributed back, we should assign a distance value from the source to each corresponding node. In fact, the distance measure procedure should be done first. At the beginning, the source will send a distance message that contains the source and the distance of the sender from the source to each direct neighbour. For the source node, the initial distance is set to 0. Every node that receives this message will check whether it has ever received such a message before. If not, it will simply set the distance of this neighbour from the source to the value that is contained in the message and its own distance from the source to this value plus 1. Otherwise, if it has already received before and the new value is smaller, it will update the distance of this neighbour from the source to the smaller value. If this new value plus 1 is smaller than its distance from the source, it will update this value correspondingly.

To complete the distance measure process in time, we should limit the maximum distance that a message can be relayed. When a node receives such messages, if it has to update its distance from the source (either receiving for the first time or receiving a shorter distance) and the distance contained in the message does not exceed the predefined maximum value, it will send new distance messages to all its neighbours telling them this information. Otherwise it will simply ignore such messages. The fact that the distance measure procedure will only have been executed once for a node when it is needed and the result can be cached for a certain time for future use should be addressed.

When the distance measure has been completed, every nearby node will know the distance of itself and its neighbours from the source; thus, the scheduling procedure can start. Suppose a number of workflow instances have been submitted to the super node, it will calculate the proportion that will be accepted and the proportion that will be distributed to other neighbours based on its own load value and the OAL value of its neighbours.

It should be addressed that this algorithm considers only itself and those neighbours whose distances from the source are greater than its own distance. The procedure is similar to the Greedy algorithm and always seeks the node with the smallest value (load value for itself and OAL value

Table I. OAL value table.

	A	B	C	D	E	F	G
A	20	50	—	53	65	—	—
B	50	40	50	—	—	52	—
C	—	47	30	50	—	—	—
D	37	—	50	80	—	—	65
E	47	—	—	—	50	55	—
F	—	30	50	—	35	80	65
G	—	—	—	43	—	50	50

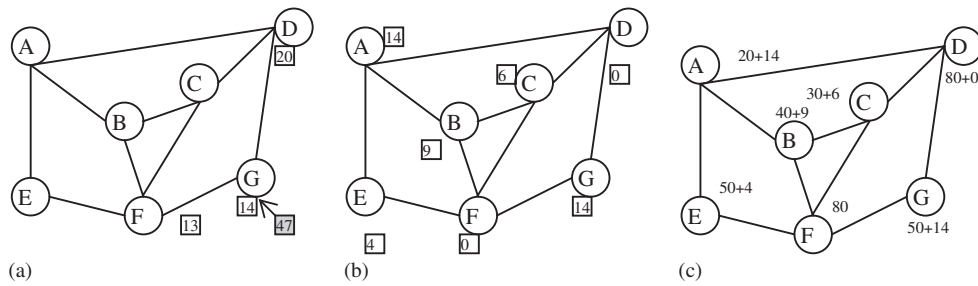


Figure 4. Scheduling result of the OAL algorithm.

for neighbours), then assigns an instance to this node and modifies the load value until all tasks are distributed. Finally, the super node will send the result to those neighbours who have instances being distributed on.

For each node, if a number of instances have been distributed on itself, it will send a request message to the source node to inform how many instances it wishes to receive. Only after the source node received this message, will the real instance transportation begin.

The scheduling result of the OAL algorithm is shown in Figure 4. It can be seen that the overall load is much more balanced than that of the original Gossip algorithm.

4.2. EMM scheduling algorithm

At the bottom level, we should design an algorithm called EMM for task scheduling with the main purpose of achieving maximum throughput in the group. The algorithm is executed periodically to provide a dynamic scheduling in a batch mode. It extends the original Min–Min algorithm and is more suitable for transaction-intensive workflows. First, we introduce two data structures that will be used for the algorithm. We depict them in Figure 5.

In workflow systems, resources are needed to execute tasks, and for every resource R_k , it can only be occupied by a task at one time. Therefore, we can allocate resources to tasks by assigning time slots to the selected tasks. As shown on the top of Figure 5, when it is decided to schedule a task $I_i T_j$ (i.e. the j th task of the i th workflow instance) on R_k , we will assign a suitable time slot of this resource to it. Of course, the data dependency and control dependency should be preserved simultaneously.

As shown at the bottom of Figure 5, the element e_{ij} in matrix e means the estimated execution time for task i on resource j . If its value equals to -1 , it means that task i cannot be executed on resource j . The original values of this matrix are estimated, but we can modify the values at runtime according to the history execution record for better estimation later.

The steps of the algorithm are described as follows:

- (1) Select ready tasks from submitted instances: A ready task is the start task or a task such as whose predecessors all have been done if the 'join' condition of the task is 'and' or one of its predecessors has been done if the 'join' condition of the task is 'or'. It should be addressed that instances are traversed in the order of their submittal time, which guarantees the fact that tasks that came earlier have higher priority to be scheduled than those that came later. Thus, the execution time of individual instances can be as minimized as possible.

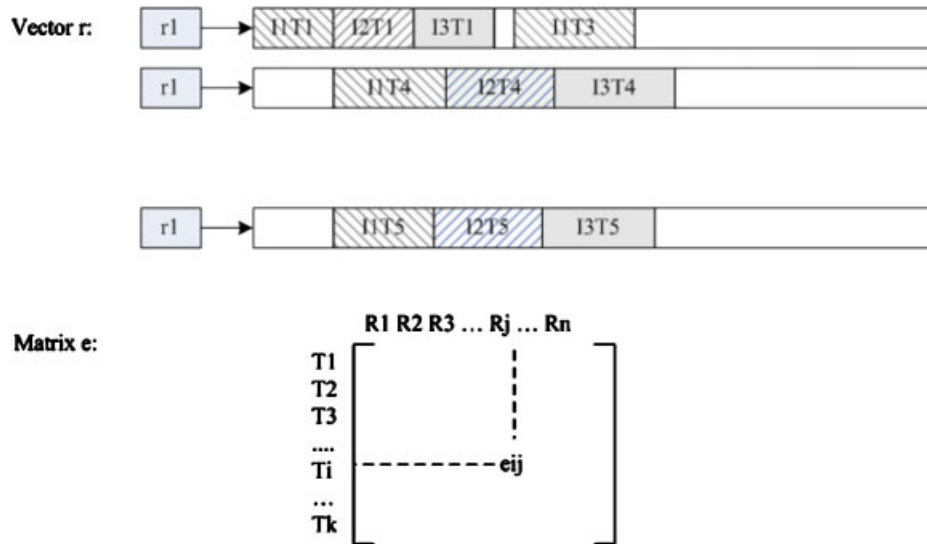


Figure 5. Data structure used in the EMM algorithm.

- (2) Compute the earliest start time (EST) on each capable resource for each task while considering data dependency and control dependency. The EST of a task is defined as follows: If the join condition of the task is 'and', the EST is equal to the latest completion time of its predecessors; if the join condition of the task is 'or', the EST is equal to the completion time of its earliest finished predecessor.
- (3) Schedule the task with the minimum EST of all tasks to the resource that is expected to complete it at the earliest time. Then the algorithm will update the data of the corresponding resource.
- (4) Go to the second step to schedule the remaining tasks until all ready tasks are scheduled.
- (5) Wait until the next scheduling period comes. During this time, new instances can be added and some unready tasks may become ready because of the completion of certain tasks.

Compared with the original Min–Min algorithm, this algorithm has its particular advantages. We can use a simple case to demonstrate the advantages of the usage of time slots. Suppose there are two tasks that need to be scheduled concurrently, we will analyse the two algorithms as follows. The Min–Min algorithm uses a Completion Matrix for the calculation of earliest completion time (ECT), and once a task has been allocated to a resource, the ECT will be updated accordingly. After instance 1 has been completed, the next available time of resource 2 will be set to the completion time of I_1T_2 . Then instance 2 will be scheduled based on the new matrix. The scheduling results are shown in Figure 6. For the EMM algorithm, the time slots before I_1T_2 are still available after instance 1 has been scheduled and can be allocated to I_2T_1 for execution.

5. COMPARISON AND SIMULATION

In this section, we first introduce four criteria for overall comparison and then perform an experimental simulation in our real-world grid workflow management system named SwinDeW-G [1].

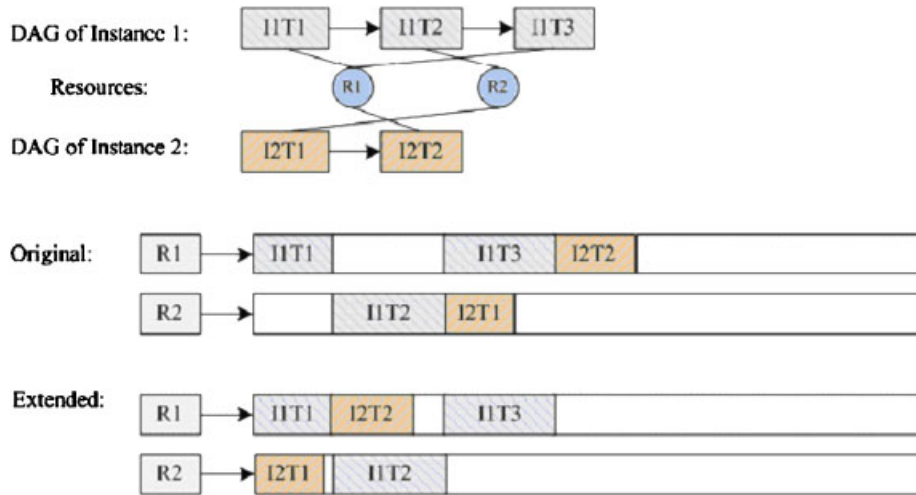


Figure 6. Example of scheduling results of original and Extended Min–Min algorithms.

SwinDeW-G is p2p-based grid workflow system. This uniqueness of the design grants it tremendous advantages of both p2p and grid technology. Our aim is to perform the original Gossip algorithm, the OAL algorithm, original Min–Min algorithm and EMM algorithm on SwinDeW-G in order to demonstrate the advantages of the OAL and EMM algorithms.

5.1. Criteria for overall comparison

For transaction-intensive workflow processing, we can compare the performance of different algorithms from four criteria as follows: the overall load balance, the mean execution time[‡], the resource utilization rate and the overall throughput. In Section 4, we have demonstrated in theory the reason why the OAL and the EMM algorithms are better in these criteria than the original Gossip algorithm and the original Min–Min algorithm. In the following section, we will use experiments to show that the OAL algorithm produces more balanced load than the original Gossip algorithm, and the EMM algorithm leads to shorter mean execution time, higher resource utilization rate and higher overall throughput than the original Min–Min algorithm in practice.

5.2. Simulation

5.2.1. Simulation environment

SwinDeW-G runs on a grid environment named SwinGrid [23]. An overall picture of SwinGrid is depicted in Figure 7, which contains many grid nodes distributed in different places. Each grid node

[‡]This intuitive criterion is used by many algorithms for performance comparison and thus is added to explicitly show the advantage resulted from high resource utilization rate of the EMM algorithm.

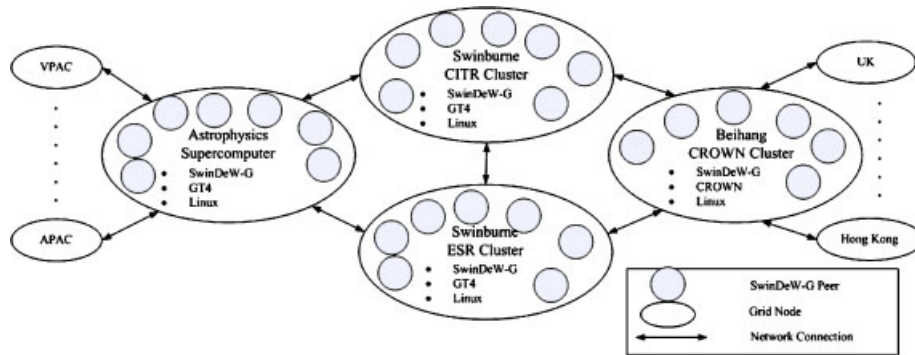


Figure 7. SwinGrid environment.

contains many computers including high-performance PCs and/or supercomputers composed of a significant number of computing units. The primary hosting nodes include the Swinburne Centre for Information Technology Research Node, the Swinburne Enterprise Systems Research laboratory Node, the Swinburne Astrophysics Supercomputer Node and the Beihang CROWN (China R&D environment Over Wide-area Network) [24] Node in China. They run Linux, Globus Toolkit (GT) 4.04 or CROWN grid toolkit 2.5 where CROWN is an extension of GT 4.04 with more middleware and hence is compatible with GT 4.04. Furthermore, the CROWN Node is also connected to some other nodes such as those in the Hong Kong University of Science and Technology and the University of Leeds in the U.K. The Swinburne Astrophysics Supercomputer Node cooperates with Australian Partnership for Advanced Computing, Victorian Partnership for Advanced Computing and so on. Currently, SwinDeW-G is deployed at all primary hosting nodes. Every cluster in Figure 7 is a scheduling group on the lower layer, and each cluster has a designated node as the super node, whereas all super nodes constitute a mesh on the upper layer.

5.2.2. Simulation process

In order to simulate the transaction-intensive scenario, we use a workflow instance generator that creates 1000 instances per second. It should be addressed that we set the initial load of every group to a random value; hence, the initial overall load is usually unbalanced.

In the simulation, we first implement the OAL and the original Gossip algorithms on SwinDeW-G, then implement the EMM algorithm and the original Min–Min algorithm and compare them based on different criteria, respectively. The criteria are shown below.

For the OAL and the original Gossip algorithms, the criterion is the variance deviation. The variance deviation of the loads of all groups can be defined as follows:

$$\sigma^2 = \frac{\sum (x - \bar{x})^2}{n}$$

where \bar{x} is the mean load value, n is the number of groups and x stands for each load value of each group in turn. This value represents how far the observations deviate from the mean. The smaller this value is, the more balanced the system is.



As for the EMM algorithm, we use about 10 000 different workflow instances that contain roughly 100 000 tasks to test the performance. The comparison criteria include the mean execution time, the resource utilization rate and the overall throughput. The shorter the mean execution time, the higher the resource utilization rate; and the higher the overall throughput, the better the algorithm.

5.2.3. Simulation results and analysis

Figure 8 shows the variance deviation of load values of the original Gossip and the OAL algorithms. The horizontal axis is the number of instances we put into simulation, and the vertical axis is the variance deviation of load values of the original Gossip and the OAL algorithms. We can see that the variance deviation of load values of the OAL algorithm is much smaller than that of the original Gossip algorithm, which shows that the OAL algorithm can produce more balanced scheduling results.

In Figure 9, the horizontal axis is the number of instances we put into simulation, and the vertical axis is the mean execution time of all instances. We can see that the mean execution time of the EMM algorithm is less than that of the original Min–Min algorithm.

In Figure 10, the horizontal axis is the number of instances we put into simulation. The vertical axis is the average resource utilization rate in percentage of all resources. It can be seen that the resource utilization rate of the EMM algorithm is higher than that of the original Min–Min algorithm.

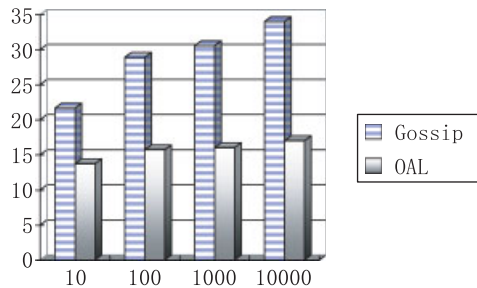


Figure 8. Variance deviation of load values of the original Gossip and the OAL algorithms.

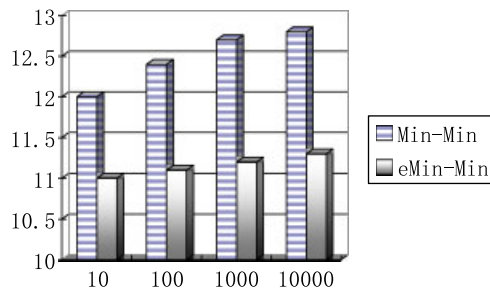


Figure 9. Mean execution time of the original and the Extended Min–Min algorithms.

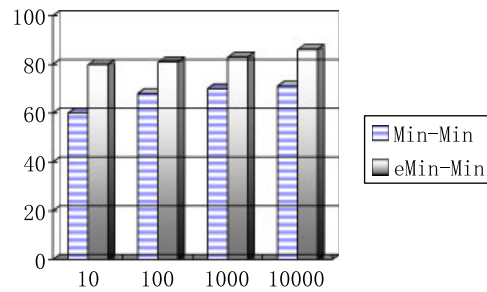


Figure 10. Resource utilization rate of the original and the Extended Min–Min algorithms.

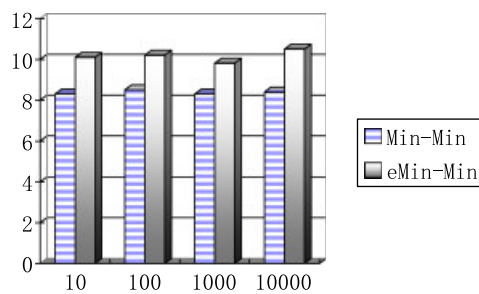


Figure 11. Overall throughput of the original and the Extended Min–Min algorithms.

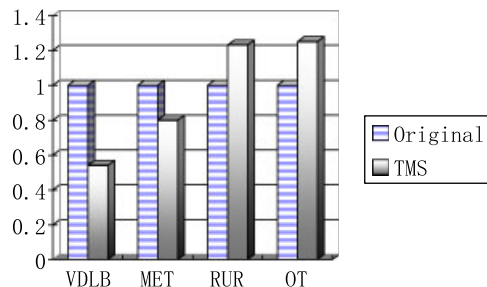


Figure 12. Overall advantage of the proposed throughput maximization strategy.

In Figure 11, the horizontal axis is the number of instances we put into simulation, and the vertical axis is the number of instances completed per time unit. When it comes to the overall throughput in a certain time, it can be seen from Figure 11 that the overall throughput of the EMM algorithm is higher than that of the original Min–Min algorithm.

To present an overall comparison based on the above analysis in Figure 12 we summarize the overall advantage of the proposed TMS from all aspects mentioned above. In this figure, variance deviation of load balance values (VDLB), mean execution time (MET), resource utilization rate (RUR), and overall throughput (OT) represent the variance deviation of load values, the mean



execution time, the resource utilization rate and the overall throughput, respectively. For each aspect, we assume that the performance of the original algorithms is 1 and compute the relative performance of the proposed TMS accordingly.

From Figure 12, we can tell that the OAL algorithm is about two times balanced than the original Gossip algorithm, and the EMM algorithm reduces approximately 20% less execution time and has roughly 20% higher resource utilization rate than the original Min–Min algorithm and thus has about 25% higher overall throughput than the original Min–Min algorithm.

6. CONCLUSIONS AND FUTURE WORK

The grid workflow systems in support of e-business require dedicated scheduling algorithms for scheduling transaction-intensive workflows. Because of the large amount of concurrent instances, i.e. transaction intensive, it is desirable to maximize the overall throughput to provide better quality of service. However, most of the existing scheduling algorithms are designed to schedule complex scientific workflows. Therefore, it is necessary to develop new algorithms for scheduling transaction-intensive workflows.

To address this problem, we have proposed a throughput maximization strategy (TMS), which contains two specific scheduling algorithms on two different layers for scheduling transaction-intensive workflows. The two algorithms are Opposite Average Load for instance scheduling and the Extended Min–Min for task scheduling. The first algorithm is to achieve the overall load balance in order to build a strong basis for the second step. The second algorithm is to further maximize the throughput by efficient management of resources. We have explained that these two algorithms are better than their respective original algorithms in theory. The final comparison and simulation have practically demonstrated that the two algorithms can produce better scheduling results than their respective original algorithms in practice when it comes to the scheduling of transaction-intensive workflows.

With these contributions, we can further investigate how to apply the TMS to real-world applications such as bank transaction and insurance claim processing systems.

ACKNOWLEDGEMENTS

The authors are grateful for the valuable comments from the anonymous reviewers, the prototyping work from our SwinDeW-G team and proofreading from S. Hunter.

REFERENCES

1. Yang Y, Liu K, Chen J, Lignier J, Jin H. Peer-to-peer based grid workflow runtime environment of SwinDeW-G. *Proceedings of the Third IEEE International Conference on e-Science and Grid Computing (e-Science07)*, Bangalore, India, December 2007; 51–58.
2. Dong F, Akl SG. Scheduling algorithms for grid computing: State of the art and open problems. *Technical Report No. 2006-504*, Queen's University, Canada, 2006; 55. Available at: <http://www.cs.queensu.ca/TechReports/Reports/2006-504.pdf> [30 October 2007].



3. Yu J, Buyya R. Workflow scheduling algorithms for grid computing. *Technical Report, GRIDS-TR-2007-10*, Grid Computing and Distributed Systems Laboratory. The University of Melbourne, Australia, 31 May 2007. Available at: <http://gridbus.csse.unimelb.edu.au/reports/WorkflowSchedulingAlgs2007.pdf> [30 October 2007].
4. Fox GC, Gannon D. Workflow in grid systems. *Concurrency and Computation: Practice and Experience* 2006; **18**(10):1009–1019.
5. Braun TD, Siegel HJ, Beck N. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. *Journal of Parallel and Distributed Computing* 2001; **61**:810–837.
6. Feo TA, Resende MGC. Greedy randomized adaptive search procedures. *Journal of Global Optimization* 1995; **6**:109–133.
7. Ludcher B, Altintas I, Berkley C, Higgins D, Jaeger-Frank E, Jones M, Lee E, Tao J, Zhao Y. Scientific workflow management and the Kepler system. *Concurrency and Computation: Practice and Experience* 2006; **18**(10):1039–1065.
8. Topcuoglu H, Hariri S, Wu MY. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Transactions on Parallel and Distributed Systems* 2002; **13**(3):260–274.
9. Fahringer T, Jugravu A, Pllana S, Prodan R, Slovis C Jr, Truong HL. ASKALON: A tool set for cluster and Grid computing. *Concurrency and Computation: Practice and Experience* 2005; **17**:143–169.
10. Sakellariou R, Zhao H, Tsiakkouri E, Dikaiakos MD. Scheduling workflows with budget constraints. *The CoreGRID Workshop on Integrated Research in Grid Computing*, Gortlach S, Danelutto M (eds.). *Technical Report TR-05-22*, University of Pisa, Dipartimento Di Informatica, Pisa, Italy, November 2005; 347–357.
11. Tannenbaum T, Wright D, Miller K, Livny M. Condor—A distributed job scheduler. *Computing with Linux*. MIT Press: MA, U.S.A., 2002.
12. Sakellariou R, Zhao H. A hybrid heuristic for DAG scheduling on heterogeneous systems. *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS'04)*, Santa Fe, New Mexico, U.S.A., April 2004; 111–123.
13. Bajaj R, Agrawal DP. Improving scheduling of tasks in a heterogeneous environment. *IEEE Transactions on Parallel and Distributed Systems* 2004; **15**(2):107–118.
14. Menasc DA, Casalicchio E. A framework for resource allocation in Grid computing. *Proceedings of the 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS'04)*, Volendam, The Netherlands, 2004; 259–267.
15. Yu J, Buyya R, Tham CK. A cost-based scheduling of scientific workflow applications on utility grids. *Proceedings of the First IEEE International Conference on e-Science and Grid Computing*, Melbourne, Australia, December 2005; 140–147.
16. Dogan A, Özgüner F. Genetic algorithm based scheduling of meta-tasks with stochastic execution times in heterogeneous computing systems. *Cluster Computing*, vol. 7. Kluwer Academic Publishers: Netherlands, 2004; 177–190.
17. Yu J, Buyya R. A novel architecture for realizing Grid workflow using tuple spaces. *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (Grid 2004)*, Pittsburgh, U.S.A. IEEE Computer Society Press: Los Alamitos, CA, U.S.A., November 2004; 119–128.
18. He L, Jarvis SA, Spooner DP, Bacigalupo D, Tan G, Nudd GR. Mapping DAG-based applications to multiclustres with background workload. *Proceedings of the IEEE International Symposium on Cluster Computing and the Grid (CCGrid'05)*, May 2005; 855–862.
19. Yang L, Schopf JM, Foster I. Conservative scheduling: Using predicted variance to improve scheduling decisions in dynamic environments. *Proceedings of the ACM/IEEE SuperComputing Conference*, Phoenix, Arizona, U.S.A., November 2003; 31–46.
20. Jin H, Shi X, Qiang W, Zou D. An adaptive meta-scheduler for data-intensive applications. *International Journal of Grid and Utility Computing* 2005; **1**(1):32–37.
21. Sakellariou R, Zhao H, Tsiakkouri E, Dikaiakos M. Scheduling workflows with budget constraints. *The CoreGRID Workshop on Integrated Research in Grid Computing*, Gortlach S, Danelutto M (eds.). *Technical Report TR-05-22*, University of Pisa, Dipartimento Di Informatica, Pisa, Italy, November 2005; 347–357.
22. Sun X, Wu M. Grid harvest service: A system for long-term application-level task scheduling. *Proceedings of 2003 International Parallel and Distributed Processing Symposium (IPDPS 2003)*, Nice, France, April 2003; 25–32.
23. Chen J, Yang Y. Adaptive selection of necessary and sufficient checkpoints for dynamic verification of temporal constraints in Grid workflow systems. *ACM Transactions on Autonomous and Adaptive Systems* 2007; **2**(2).
24. CROWN Team 2006. CROWN Portal, <http://www.crown.org.cn/en/> [30 October 2007].