Applied Mathematics & Information Sciences
An International Journal

A Scheduling algorithm for Multi-Tenants Instance-Intensive Workflows

Lizhen Cui [1,2], *Tiantian Zhang* [1,2], *Guangquan Xu* [3] *and Dong Yuan*[4]

[1]School of Computer Science and Technology, Shandong University, 250101,Jinan, China
[2]Shandong Provincial Key Laboratory of Software Engineering
[3]School of Computer Science and Technology, Tianjin University, 300072,Tianjin,China
[4]Faculty of Information and Communication Technologies at Swinburne University of Technology, Melbourne, Australia

Received: 28 Jul. 2012, Revised: 20 Sep. 2012, Accepted: 5 Nov. 2012
Published online: 1 Feb. 2013

Abstract: As a key service model in cloud computing, SaaS applications are becoming increasingly popular. Multi-tenancy is a key characteristics of SaaS applications. Business processes play a key role in SaaS applications because of the composability and reusability of software services. This paper focuses on multi-tenants instance-intensive workflows system, in which workflows have a large number of instances belonging to multiple tenants in a SaaS environment, and further proposes a scheduling algorithm for multi-tenants workflow instances. This algorithm improves the quality of service (QoS) for tenants and saves the execution cost of workflows. The simulation results demonstrate that the proposed algorithm guarantees the workflow execution conforming to the deadline set by tenants, and reduces the mean execution time for tenants in high priority whilst saves the execution cost for service providers.

Keywords: Multi-tenants, Instance-intensive workflow, scheduling algorithm

1. Introduction

As the development of cloud computing [1,2] and software technology, SaaS (Software as a Service) is widely used for bringing benefits to both software service providers and tenants. Multi-tenants [3] architecture is deemed as an important technology for SaaS application providers to achieve high profit margin. SaaS application is delivered to multiple tenants through the Internet in a single-instance multi-tenants architecture model.

The character of SaaS application can be simply summarized as multi-tenancy, scalable and customizable. Multi-tenancy allows multiple tenants to share a single application instance securely, which can effectively reduce service costs. Scalable means that the number of servers and instances can be increased or decreased according to the demand, so SaaS architecture is scalable to an arbitrary number of tenants. Customizable allows tenants to alter the user interface, change data fields shown up in the program, and turn off or on several business process functions [4].

Many SaaS applications require workflow processing in which tasks are executed based on their control or data dependencies. Workflow scheduling plays a key role of determining the efficiency of workflow system, therefore, it has a practical significance to the development of workflow scheduling algorithm for multi-tenants SaaS environment. However, there are no dedicated scheduling algorithms for multi-tenants instance-intensive workflows. This paper presents a novel scheduling algorithm which considers the new features of multi-tenants SaaS application.

In a multi-tenants environment, each tenant has a certain number of workflow instances, and with the increase of tenants, there will be a huge number of workflow instances belonging to multiple tenants, we call such workflows multi-tenants instance-intensive workflows. The multi-tenants instance-intensive workflows scheduling algorithm should consider following criterions: 1)the quality of service experience (QoSE) of tenants in different service level agreements (SLA); 2) mean execution time of multiple workflow instances; 3)save the execution cost for service providers.

To address these challenges, we propose a scheduling algorithm to efficiently support multi-tenants instance-intensive workflows. The objectives of our scheduling strategy are: (1) Workflow instances execution

* Corresponding author e-mail: clz@sdu.edu.cn

meet the deadline imposed by tenants in different priorities. (2) The workflow instances mean execution time of high priority tenants is as short as possible, which means high user QoSE. (3) Minimize the execution cost for service providers.

The remainder of the paper is organized as follows: In Section2, we present an overview of the related work. In Section 3, we formulate the problem of scheduling instance-intensive workflows in a multi-tenants environment. Section 4 discusses the details of the scheduling algorithm. Experimental details and simulation results are presented in Section 5. Finally, we conclude the paper and point out the future work in Section 6.

2. Related Work

In SaaS model, service providers provide collaborative process control application through workflow technology, and meanwhile the workflow scheduling algorithm determines the efficiency of the workflow system, therefore, a new algorithm for multi-tenants instance-intensive workflows is very important.

Scheduling workflow tasks onto a set of available machines in distributed systems is a well-know NP-complete problem, even in the simplest form [6,7].

Some workflow scheduling algorithms mainly target on the minimization of execution, where typical grid workflow scheduling algorithms can be classified to this type, such as the HEFT heuristic[8] used by ASKALON [9], Myopic heuristic [10] used by grid Condor DAGMan [11], Min-Min Heuristic [12] and GRASP meta-heuristic [13] used by Kepler [14], and so on. However, the algorithms mentioned above do not consider other factors such as monetary cost of accessing resources and various users' QoS. As a result it is not practical to apply such algorithms to a multi-tenants environment.

In contrast, some workflow scheduling algorithms attempt to consider the constraint of monetary cost as the major factor. The existing representative algorithms include Genetic meta-heuristic implemented in GridBus [15], Dead line-MDP heuristic [16], Back-Tracking heuristic [17], Loss and Gain [18], and so on. However these algorithms are only designed for scheduling a single workflow instance.

In [5], the CTC algorithm is designed to schedule instance-intensive workflow on a cloud workflow system SwinDew-C [19]. In [20], a throughput maximization strategy is proposed for scheduling transaction intensive workflows. However, such instance-intensive workflow scheduling algorithms have not considered the multi-tenants environment. In [22], the authors present the parallel real-time scheduling algorithm on multi-core platform. Dynamic programming and multi-objective linear programming approaches [23], genetic ant algorithm [24], a learning based evolutionary approach [25] and localization algorithm [26] are also effective optimization algorithms, but they are not suitable for the workflow scheduling problem.

3. Problem Formulation

Before the description of the scheduling strategy, we first give some definitions which will be used late.

Definition 1: (Multi-Tenants Workflows (MTW)) Multi-tenants Workflows is defined as a triple $MTW = (Ten, WF, OD)$, where

(1) $Ten = (ID, lev)$ is the tenant, where ID is tenant ID, lev is the priority of tenant. $Ten \in Tens$, $Tens = \{Ten_1, Ten_2, \ldots, Ten_m\}$, where m is the number of tenants.

(2) $WF = (T_s, E_s)$ is the workflow model of a tenant. WF can be described as a directed acyclic graph, where T_s is a finite set of workflow tasks $T_i (1 \leq i \leq n)$, E_s is the set of directed arcs (T_i,T_j). We further define that T_i is the parent task of T_j and T_j is the child task of T_i. We assume that a child task cannot be executed until all its parent tasks are completed. In a workflow graph, we call a task which has no parent task an entry task denoted as T_{entry} and a task which has no child task an exit task denoted as T_{exit}.

(3) OD is the time constrained (i.e. overall deadline) by the tenant for workflow execution.

A workflow model of the ith tenant is described as $MTW_i = (Ten_i, WF, OD_i)$, where Ten_i is the ith tenant, $WF = (T_s, E_s)$ is the workflow of the ith tenant, and OD_i is the deadline of the ith tenant.

From the above definitions, we know that different tenants have the same workflow model, but different deadlines.

Definition 2: (Tenant Workflow Instance) $I = \{I_1, I_2, \ldots, I_m\}$ is the set of workflow instances of all tenants, where I_i is the set workflow instances of ith tenant and m is the total number of tenants. $I_i = \{I_{i1}, I_{i2}, \ldots, I_{io}\}$, I_{ij} is the jth workflow instance of ith tenant, $0 \leq j \leq o_i$, o_i is the total number of instances of ith tenant.

Definition 3: (Tenant Workflow task instance) T_{ijk} is the kth task of jth instance of ith tenant, $1 \leq i \leq m$,$0 \leq j \leq o_i$,$0 \leq k \leq n$, where m is the total number of tenant, o_i is the total number of instances of ith tenant, and n is the number of the tasks in the workflow model.

Definition 4: (Tenant Workflow Instance Time-Saving-Degree (TSD)) $TSD = 1 - \frac{Makespan+SWT}{OD}$, where $Makespan$ (execution time) is the time spent from the beginning of the first task to the end of the last task for a workflow instance execution, SWT (scheduling waiting time) is the time spent from the workflow instance submission to the beginning of the first task, OD is the time constrained by the tenant for workflow execution.

Definition 5: (Sub-Deadline (SD)) The latest completion time allocated to a single task. $SD_{T_{ijk}}$ is the sub-deadline of task T_{ijk}.

Definition 6: (Service Resource (R)) Software service resources are used to execute workflow tasks. $SetR = \{SetR_1, SetR_2, \ldots, SetR_n\}$ is the set of various type resources. $SetR_k = \{R_k^1, R_k^2, \ldots, R_k^{p_k}\}$ is the set of resource, every resource in $SetR_k$ is capable of executing the task T_k $(1 \leq k \leq n)$, but only one resource can be assigned for the execution of a task, n is the total number of workflow task, p_k is the total number of resources which is capable for executing the task T_k ,R_k^h is the hth resource which is capable for executing task T_k.

Resources have different processing capabilities delivered at different prices. Every resource can execute one task at the same time. The scheduling problem is to select a suitable resource R_k^h from $SetR_k$ for task T_k.

Definition 7: (Execution Time (ET)) $ET_{T_{ijk}}^{R_k^h}$ is the execution time for executing task T_{ijk} on resourceR_k^h .

Definition 8: (Execution Cost (EC)) $EC_{T_{ijk}}^{R_k^h}$ is the execution cost for executing task T_{ijk} on resource R_k^h. Execution cost is the money spent of the service provider.

Definition 9: (Earliest Start Time (EST)) The earliest start execution time of task T_{ijk} is denoted as $EST_{T_{ijk}}$. EST of a task is determined by the completion time of its predecessors.

4. Multi-tenants Instance-intensive Workflows Scheduling Algorithm

Service providers are increasingly concerned of users' QoSE and how to save cost for executing multi-tenants instance-intensive workflows. Considering the new features of multi-tenants instance-intensive workflows, we propose a Time-Saving-Degree and cost optimization (TSD-Cost) scheduling algorithm which mainly focuses on increasing users' QoSE and minimizing execution cost for service providers. In this paper, QoSE can be described as the higher the tenant's priority is, the bigger the mean Time-Saving-Degree is.

Before we give the details of the schedule strategy, we first give the overview of the algorithm and present the algorithm details next. The algorithm can be divided into 6 steps:

Step1. Calculate the sub-deadline for tasks of all instances.

Step2. Select ready tasks and group tasks according to the resources required by every task, and sort the tasks by the sub-deadline and the tenant level.

Step3. Sort the resources according to the execution cost.

Step4. Allocate a suitable time-slot of resource to every ready task.

Step5. For each task allocated on a resource, adjust the task start execution time.

Step6. Go to step2 for the next round scheduling.

The details of each step are provided in the following sub sections, and the pseudo-code of the schedule algorithm is given below.

Input: I : Set of workflow instances sets of each tenant
Output: $Schdule$: A schedule

1.$Schdule \leftarrow \varphi$;
//Step1:Calculate the sub-deadline for tasks of all instances
2.**for** each $I_i \in I$
3. Calculate the sub-deadline according to the principle introduced below
//Step2: Group and sort ready tasks
4.$TaskQueue[] \leftarrow GroupTasks(I)$
//Step3: Sort the resources.
5.$ResourceQueue[] \leftarrow SortResources(TaskQueue[])$
//Step4: Assign a suitable time slot of resource to each ready task.
6.$partialSchdule \leftarrow AssingResources(TaskQueue[], ResourceQueue[])$
//Step5:Adjust task start execution time
7.$partialSchdule \leftarrow ASET(partialSchdule)$
8.$Schdule \leftarrow Schdule + partialSchdule$
//Step6: Go to step2 for the next schedule
9.Go to step2 for the next schedule.

Algorithm 1: Scheduling Algorithm

Step1: Calculate the sub-deadline

Similar to the overall deadline of the each workflow instance, a sub-deadline is the latest completion time allocated to each task instance, which the completion of the task should not exceed

The Deadline-MDP heuristic [16] shows a practical method to divide the overall deadline into sub-deadline, which contains two major steps: in the first step it categorized the tasks into simple tasks and synchronization tasks.

In the second step, the overall deadline is distributed to each task in proportion to their minimum processing time. In this paper, we reference the Deadline-MDP to divide the overall deadline. The time division complies with the following principles:

1. The cumulative sub-deadline of any independent path between two synchronization tasks must be same.

2. The cumulative deadline of any path from T_{entry} to T_{exit} is equal to the overall deadline.

3. Any assigned sub-deadline must be greater than or equal to the minimum processing time of the corresponding task.

4. The overall deadline is divided over tasks in proportion to their mean execution time. The mean execution time of a task is the mean task execution time on every available resource

Step2: Group and sort ready tasks

The second step of the scheduling algorithm is to group and sort ready tasks. Ready tasks are selected from

the multiple instances of multiple tenants with the same workflow model. A ready task is the start task of a workflow instance or a task whose predecessors have all been allocated. The ready tasks with the same resource requirements are grouped and allocated together, while different task groups can be scheduled in parallel. Every task in the same task group has the same task type, e.g. all task instances T_1 will be grouped to TG_1.

After ready tasks have been selected and grouped as task groups, for each task group, sort tasks by the sub-deadline in descending order and then sort the tasks by the tenant level in ascending order.

The purpose of sorting is to first select the task with the maximum sub-deadline and minimum tenant level for scheduling. The pseudo-code of this period is given below:

Input: I : Set of workflow instances sets of each tenant
Output: $TaskQueue[]$: Array of sorted task queues
//Group and sort ready tasks
// sub-step1 Group ready tasks

1.$TaskQueue[] \leftarrow \varphi$;
2.**for** each $I_i \in I$ do
3. **for** each $I_{ij} \in I_i$ do
4. **while** $\exists T_{ijk} \in I_{ij}$ is ready
5. $ReadyTasks \leftarrow T_{ijk}$
6.**for** each task T_{ijk} in $ReadyTasks$ do
7.$TG_k \leftarrow T_{ijk}$ // T_{ijk} requires resource R_k^h
//sub-step2 Sort ready tasks
8.**for** each task group TG_k do
9. **while** $\exists T_{ijk} \in TG_k$ do
10. insert task T_{ijk} into queue $TaskQueue[k]$ according to the strategy introduced above
11.**return** array of sorted task queues $TaskQueue[]$

Algorithm 2: Group Tasks (GroupTasks)

Step 3: Sort resources according to the execution cost

There are many resources can be used to executing a same type task. Different resources will have different executing speed and cost. In general, higher executing speed which can reduce the execution time normally results in higher execution cost. In this step, every resource in the same resource set will be sorted by their executing cost from small to large. The purpose of sorting is to first select the resource with the lowest execution cost for a ready task. The pseudo-code of this period is given below:

Step 4: Allocate resource time-slot for every ready task

This step is to map ready tasks onto resources. We take resource as a time shaft, for each resource R, we can use a time-slot window to represent the time allocation status of its execution unit. The window represents the time used for task execution on the resource. Figure 1 shows the overview of the allocating strategy.

Input: $TaskQueue[]$: array of sorted task queues
Output: $ResourceQueue[]$: Array of sorted resource queues

1.$ResourceQueue[] \leftarrow \varphi$
2.**for** each $TaskQueue[k] \in ResourceQueue[]$
3. **for each** $R_k^h \in SetR_k$ do
4. insert R_k^h into queue $ResourceQueue[k]$ by cost in ascending order
5.**return** $ResourceQueue[]$

Algorithm 3: Sort Resources(SortResources)

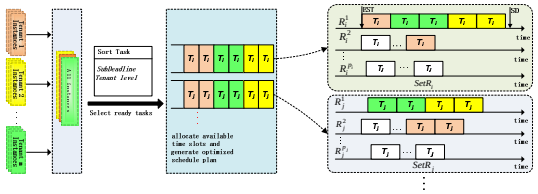


Figure 1 An Overview of Scheduling Strategy

The pseudo-code of this period is given below:

Step 5: adjust start execution time

According to our allocating strategy, the start execution time of task on resource may be later than the earliest start time of task, so after allocating tasks, we should adjust start execution time of every task on each resource.

The related terms defined are listed below:

$Start_T^R$: It is the start execution time of task T on resource R.

End_T^R : It is the completion time of task T on resource R.

The pseudo-code of this period is given below:

Step6. Next round scheduling

After all ready tasks being allocated to specific resources, the scheduler will go to step 2 for next scheduling until all tasks are allocated.

5. Simulation and Comparison

In order to evaluate the performance of the schedule algorithm, we have developed an experimental simulator. We compare our proposed scheduling algorithm denoted as TSD-Cost with Deadline-MDP algorithm which is more effective than Deadline-Level and Greedy-Cost algorithms which are derived from the cost optimization algorithm in Nimrod-G [21]. The Deadline-MDP approach is initially designed for scheduling a single scientific workflow on Grids.

Workflow model simulation

As execution requirements for tasks in SaaS workflow application are heterogeneous, service type is used to

Input: $TaskQueue[]$ $ResourceQueue[]$
Output: $partialSchdule$: A partial schedule of I

1.$partialSchdule \leftarrow \varphi$
2.**for** each $TaskQueue[k]$ is not empty
3. $T \leftarrow$ first task in $TaskQueue[k]$
4. **while** $ResourceQueue[k]$ is not empty
5. $R \leftarrow$ first Resource in $TaskQueue[k]$
6. **if**$(EST_R = -1)$
7. **if**$((SD_T - ET_T^R) \geq EST_T)$
8. $partialSchdule \leftarrow partialSchdule + assign(R,T,SD_T - ET_T^R, SD_T)$
9. $EST_R = SD_T - ET_T^R$
10. **else**
11. $R \leftarrow$ next Resource in $ResourceQueue[k]$
12. **else**
13. **if**$(SD_T \geq EST_R \&\& (EST_R - ET_T^R) \geq EST_T)$
14. $partialSchdule \leftarrow partialSchdule + assign(R,T,EST_R - ET_T^R, EST_R)$
15. $EST_R = EST_R - ET_T^R$
16. **elseif**
$(((SD_T < EST_R) \&\& (SD_T - ET_T^R)) \geq EST_T)$
17. $partialSchdule \leftarrow partialSchdule + assign(R,T,SD_T - ET_T^R, SD_T)$
18. $EST_R = SD_T - ET_T^R$
19. **else**
20. $R \leftarrow$ next Resource in $TaskQueue[k]$
21.**return** $partialSchdule$

Algorithm 4: AssignResources(AssingResources)

Input: $partialSchdule$: A partial shedule
Output: $partialSchdule'$: A partial shedule with start execution time adjusted

1.**for** each R in $Schdule.assign(R,-,-,-)$
2. **for** each task T allocated on R
3. insert task T into queue $TaskQueue_R$ according to the $Start_T^R$ from small to large
4. $end \leftarrow \infty$
5. **while** $TaskQueue_R$ is not empty
6. $T \leftarrow$ first task in $TaskQueue_R$
7. **if** $(Start_T^R > EST_T \&\& end == \infty)$
8. $Start_T^R \leftarrow EST_T$
9. $End_T^R \leftarrow Start_T^R + ET_T^R$
10. **if** $(Start_T^R > EST_T \&\& EST_T \geq end)$
11. $Start_T^R \leftarrow EST_T$
12. $End_T^R \leftarrow Start_T^R + ET_T^R$
13. **if** $(Start_T^R > EST_T \&\& EST_T < end)$
14. $Start_T^R \leftarrow end$
15. $End_T^R \leftarrow Start_T^R + ET_T^R$
16. $end \leftarrow End_T^R$
17.**return** $partialSchdule'$

Algorithm 5: Adjust Start Execution Time(ASET)

represent different type of resource. In our experiment environment, each task requires a certain type of service. We randomly generate various workflow graphs with only 3-9 tasks which represent multi-tenants instance-intensive workflows and randomly select a workflow model for our test.

Workflow instance simulation

In order to test the performance of different algorithm, a workflow instance generator creates 10,000-18,000 instances are used to simulate the huge number of workflow instances. These instances belong to 100 tenants whom can be belonged to 3 levels. Each tenant has 100-180 instances. In this experiment, tenant level is simply determined by the deadline (i.e., the greater the deadline is, the lower tenant level is). For example, a tenant with deadline in [100, 200] belong to level 1, a tenant with deadline in [50, 100] belong to level 2.

Multi-tenant environment simulation

According to workflow model, our simulate cloud environment has a certain number types of resources which is equal to the number of task types. Each type of resource has 3-8 resources with different execution time and execution cost.

The Figure 2 compare the execution cost of using TSV-Cost and Deadline-MDP for scheduling a same batch of workflow instances with instances number 10000, 12000, 14000, 16000, and 18000 respectively.

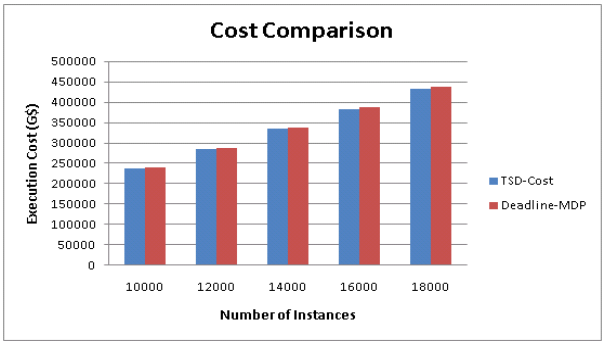
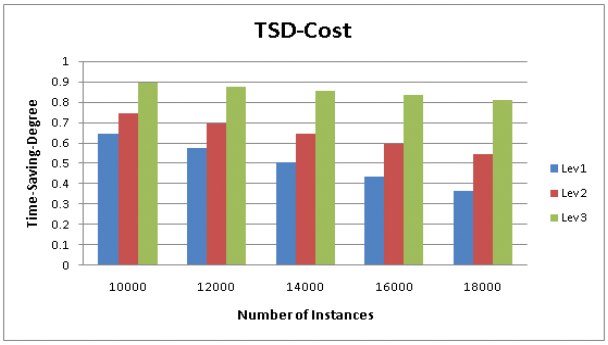


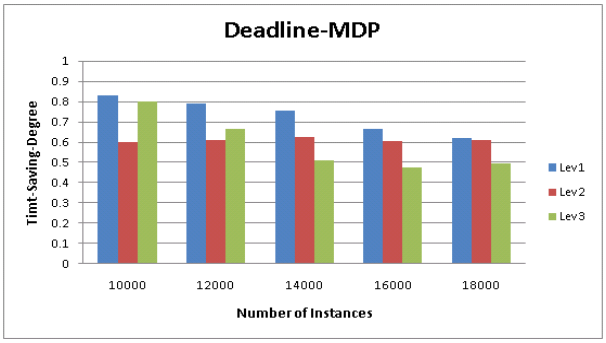
Figure 2 Comparison on execution cost

Both algorithms meet the deadline, but the execution cost of TSD-Cost algorithm is less than that of the Deadline-MDP algorithm although slightly, which mean TSD-Cost algorithm can save more money than Deadline-MDP algorithm for service providers.

Figure 3(a) demonstrates the comparison results on the mean tenant workflow instance time-saving-degree of different levels on TSD-Cost algorithm. It can be seen that the time-saving-degree of high level tenant is always higher than that of low level tenant in all circumstances.



(a) Time-Saving-Degree of TSD-Cost algorithm



(b) Time-Saving-Degree of Deadline-MDP algorithm

Figure 3 Comparison of Time-Saving-Degree

Figure 3(b) demonstrates the comparison results on the mean tenant workflow instance time-saving-degree of different levels on Deadline-MDP algorithm. It can be seen that the time-saving-degree of high level tenant is not always higher than that of low level tenant, which means low user experience.

6. Conclusions and future work

As scheduling is always a key factor for workflow systems, it is necessary to design workflow scheduling algorithm suitable for SaaS multi-tenants environment. The primary work of this research can be divided into the following five aspects:

1. Analyze and conclude the new challenges of scheduling multi-tenants instance-intensive workflows;

2. Give the formalization description of scheduling issue.

3. Present a scheduling algorithm for multi-tenants instance-intensive workflows.

4. Introduce a case study to show the details of the scheduling process;

5. Plan the performance experiments to prove the proposed algorithm is efficient and effective.

In the future work, we plan to develop a realistic experiment environment and study how to add more QoS constrained to workflows.

Acknowledgement

The research work was supported by the National Natural Science Foundation of China under Grant No. 61003253,61 003080. Natural Science Foundation of Shandong Province of China under Grant No. ZR2010FQ010, ZR2010FM031.

References

[1] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic, “Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility”, Future Generation Computer Systems, Elsevier Science, Amsterdam, June 2009, Volume **25**, Number 6, pp. 599-616.

[2] M. Armbrust, Above the Clouds: A Berkeley View of Cloud Computing, EECS Department, University of California, Berkeley, 2009.

[3] Chang Jie Guo, Wei Sun, Ying Huang, Zhi Hu Wang, Bo Gao, “A Framework for Native Multi-Tenancy Application Development and Management”, CEC-EEE 2007, 2007, **7**, 551-558.

[4] F. Chong and G. Carraro, “Architecture strategies for catching the long tail,” MSDN Library, Microsoft Corporation, 2006.

[5] K. Liu, H. Jin, J. Chen, X. Liu, D. Yuan and Y. Yang. A Compromised-Time-Cost Scheduling Algorithm in SwinDeW-C for Instance-Intensive Cost-Constrained Workflows on Cloud Computing Platform. International Journal of High Performance Computing Applications, Sage, to appear (accepted on Jan. 18, 2010).

[6] R. L. Graham. “Bounds on Multiprocessing Timing Anomalies”. SIAM Journal of Applied Mathematics, **17(2)**: 416-429, 1969.

[7] G. M. R., J. D. S., and S. Ravi. “The Complexity of Flowshop and Jobshop Scheduling”. Mathematics of operations research, **1(2)**: 117-129, 1976. Schlumberger Besanc, on, (2001).

[8] H. Topcuoglu, S. Hariri, and M. Y. Wu. Performance-Effective and Low-Complexity Task Scheduling for Heterogeneous Computing, IEEE Transactions on Parallel and Distributed Systems, volume **13**, issue 3, pp. 260-274, 2002.

[9] T. Fahringer, A. Jugravu, S. Pllana, R. Prodan, C. Slovis Jr, and H. L. Truong. ASKALON: a tool set for cluster and Grid computing. Concurrency and Computation: Practice and Experience, volume **17**, pp. 143-169, Wiley InterScience, 2005.

[10] M. Wieczorek, R. Prodan, and T. Fahringer. Scheduling of Scienti c Work ows in the ASKALON Grid Environment. ACM SIGMOD Volume **34**, Issue 3, pp. 56-62, Sept. 2005.

[11] T. Tannenbaum et al., Condor - A Distributed Job Scheduler, Beowulf Cluster Computing with Windows Section: Managing clusters, The MIT Press, MA, USA, 2001.

[12] T. D. Braun, H. J. Siegel, and N. Beck. A comparison of eleven static heuristics for mapping a class of independent tasks onto heterogeneous distributed computing systems. Journal of Parallel and Distributed Computing, vol.**61**, issue 6, pp. 810-837, 2001.

[13] T. A. Feo and M. G. C. Resende. Greedy Randomized Adaptive Search Procedures, Journal of Global Optimization, volume **6**, pp. 109-133, 1995.

[14] B. Ludcher, I. Altintas, C. Berkley, D. Higgins, E. Jaeger-Frank, M. Jones, E. Lee, J. Tao, and Y. Zhao. Scientific Workflow Management and the Kepler System. Concurrency and Computation: Practice and Experience, volume **18**, issue 10, pp. 1039-1065, 2006.

[15] Buyya, R. and Venugopal. The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report, Proc. of 1st IEEE International Workshop on Grid Economics and Business Models (GECON 2004), pp. 19-36, Seoul, Korea, April 2004.

[16] Yu, R. Buyya, and C.K. Tham. 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, pp. 140-147, Dec 2005.

[17] D. A. Menasc and E. Casalicchio. A Framework for Resource Allocation in Grid Computing, Proc. of the 12th Annual International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems, pp. 259-267, 2004.

[18] E.Tsiakkouri et al.. Scheduling Workflows with Budget Constraints, The CoreGRID Workshop on Integrated research in Grid Computing. Technical Report TR-05-22, University of Pisa, Dipartimento Di Informatica, Pisa, Italy, Nov. 28-30, 2005, pp. 347-357.

[19] J. Yan, Y. Yang and G. K. Raikundalia, SwinDeW-a P2P-based Decentralized Workflow Management System, IEEE Transactions on Systems, Man and Cybernetics, Part A, **36(5)**, 922-935, 2006.

[20] Ke Liu, Jinjun Chen, Yun Yang and Hai Jin, "A throughput maximization strategy for scheduling transaction-intensive workflows on SwinDeW-G", Concurrency and Computation: Practice and Experience, Wiley, **20**(15):1807-1820, Oct. 2008.

[21] Buyya, R., Giddy, J. and Abramson, D. (2000). An Evaluation of Economy based Resource Trading and Scheduling on Computational Power Grids for Parameter Sweep Applications, Proc. of the 2nd Workshop on Active Middleware Services (AMS 2000), pp. 1-10, Pittsburgh, USA, August 2000.

[22] Zhou, BH (Zhou, Benhai); Qiao, JZ (Qiao, Jianzhong) Lin, SK (Lin, Shukuan). Research on Parallel Real-time Scheduling Algorithm of Hybrid Parameter Tasks on Multi-core Platform. Appl. Math. Inf. Sci. **5(2)**, pp. 211-217.

[23] Dynamic Programming and Multi Objective Linear Programming approaches, Appl. Math. Inf. Sci. vol.**5(2)**, 2011, pp. 253-263.

[24] Jinghui Gao and Rui Shan, A New Method for Modification Consistency of the Judgment Matrix Based on Genetic Ant Algorithm, Appl. Math. Inf. Sci. Vol. **6** No. 1S (2012) pp. 35S-39S.

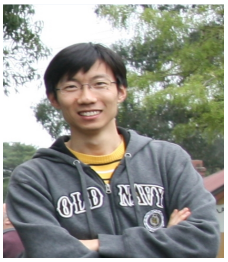
[25] Isazadeh, Ayaz; Izadkhah, Habib; Mokarram, A. H. A Learning based Evolutionary Approach for Minimization of Matrix Bandwidth Problem. Appl. Math. Inf. Sci. Vol. **6(1)** (JAN 2012) pp. 51-57.

[26] Bao, Peng. Localization Algorithm Based on Sector Scan for Mobile Wireless Sensor Networks. Appl. Math. Inf. Sci. Vol.**6** No. 1S(2012) pp.99S-103S.

Lizhen Cui received the PhD degree in computer science from the Shandong University in 2006.He is a associate professor in the School of Computer Science and Technology at the Shandong University. His research interests include service computing, workflow and data management for cloud computing.

Tiantian Zhang Graduated from Shandong normal university, China. Now she is a M.S. candidate in the School of Computer Science and Technology at the Shandong University. Her research interests include resource allocation, workflow and data management for cloud computing.

Guangquan XU is a Ph.D. and associate professor at the School of Computer Science and Technology, Tianjin University, China. He received Ph.D. degree from Tianjin University in March 2008. He is a member of the CCF and ACM. His research interests include verified software, trusted computing, trust and reputation.



Dong Yuan received the B.Eng. degree and M.Eng. degree from Shandong University, Jinan, China, in 2005 and 2008, the Ph.D degree from Swinburne University of Technology, Melbourne, Australia, in 2012, all in computer science. He is currently a research fellow in the Faculty of Information and Communication Technologies at Swinburne University of Technology, Melbourne, Australia. His research interests include data management in parallel and distributed systems, scheduling and resource management, grid and cloud computing.