QoS Constrained Workflow Scheduling Scheme Using the Colored Petri-Net Model

Yun-Gi Ha, Seong-Hwan Kim, Kyung-No Joo, Chan-Hyun Youn

Department of Electrical Engineering, KAIST
Daejeon, Korea
{milmgas, s.h_kim, eu8198, chyoun}@kaist.ac.kr

Abstract. We propose the Quality of Service (QoS) constrained workflow scheduling scheme utilizing colored Petri-Net model to apply the task division policy which enables to expand the QoS-guaranteed range, and effectively cope with resource performance variance. The proposed algorithm investigates each task's workload then decides its distribution rate. Afterwards, the proposed algorithm allocates the cheapest Virtual Machine (VM) to each task which can satisfy the subdeadline of the task. If there is no suitable VM resource, the task division policy is applied while penalty cost is considered. We compared the performance of the proposed algorithm, which is called as the Phased Workflow scheduling Scheme with Division (PWSD), with the Phased Workflow scheduling Scheme (PWS) which did not consider the division policy. The performance comparison, based on randomly generated task within the same work-flow topology, shows that the proposed scheme outperforms PWS, which means that it expands QoS-guaranteed range and enhances robustness to resource performance variance.

Keywords: Workflow scheduling, Task division, Policy-based, Cloud computing

1 Introduction

As cloud computing offers a feasible solution for workflow applications executed in the distributed environment, now users are able to take computing resources in cloud venders at pay-per-use manner without establishing their own computing infrastructure. However, there still exits issues such as resource management issues which are needed to be resolved to achieve better performance at lower cost. A workflow management system which coordinates the execution of each task by deciding resources to assign different tasks and the order of task execution in cloud environment is introduced to resolve those issues. Users make a Service Level Agreement (SLA) contract with the cloud service broker about price and Quality of Service (QoS) in order to define region for QoS-guaranteed workflow processing service [1] [2]. Therefore, it is essential for the cloud broker to consider a workflow scheduling algorithm which allocates the proper task to the proper resource so that workflow execution request can be processed by the proper resource in the proper way at the proper time, namely, in the way of satisfying QoS constraints [3].

As workflow scheduling is a NP-hard problem, there exists many researches to resolve the scheduling problem. Sakellariou, R [5] proposed two different approaches which change initial schedule if the given budget is less or greater than the cost required to process submitted workflow. Although this heuristic supports effective scheduling for a workflow, it is hard to deal with performance variance in cloud

environment as the heuristic is sorted as static scheduling. In addition, it incurs relatively long execution time because of time-consuming resource reassignment iterations. Therefore, this algorithm is not applicable in cloud environment. Kim [6] suggested workflow management scheme which schedule subtasks onto resources according to given budget or completion time. After investigating load of tasks in a workflow to decide QoS constraints distribution rate, the workflow is processed by designated scheme. As Kim [6]'s workflow scheduling scheme belongs to dynamic scheduling scheme, it can effectively cope with resource performance problem. Also, as it is a simple heuristic which doesn't require reassignment process, it instantly finds aimed schedule. However, its QoS-guaranteed service range and robustness to resource performance variance are limited with finite resource set. However, Kim [6]'s workflow scheduling scheme is providing limited service range whose QoS is guaranteed.

In this paper, we propose a QoS constraints workflow scheduling scheme which utilize task division strategy based on the colored Petri-Net model. We considered task division policy to expand QoS-guaranteed service range and improve performance variance to resource performance variance.

2 Problem Description and Related Works

As the service quality of cloud computing varies even in the same environments, it is not easy to guarantee the Service Level Agreement (SLA). As shown in **Fig. 1**, there exist performance gap for a task among the same type of resources. The performance variance is caused as each cloud resource is provided with its superficial specification, such as the number of processor core, the capacity of storage in terms of GB [12]. If the variance between the worst performance and the best performance is big, then it is much more difficult to guarantee the SLA, especially in terms of user-specified QoS constraints. This makes QoS-satisfactory scheduling even harder when tough QoS constraints are given.

There exists many workflow scheduling scheme to utilize computing resources efficiently. HEFT [4] is an algorithm that selects the task with the largest load of the slowest execution path, then it allocates the task onto the fastest processors to minimize overall workflow completion time. Tough it produces a near-optimum scheduling which minimizes completion time, it cannot guarantee SLA in performance-varying environment. Sakellariou, R [5] proposed two different approaches which change initial schedule if the given budget is less or greater than the cost required to process submitted workflow. Although this heuristic supports effective scheduling for a workflow, it is hard to deal with performance variance in cloud environment as the heuristic is kind of static scheduling. In addition, it incurs relatively long execution time because of time-consuming resource reassignment iterations. Therefore, this algorithm is not applicable in cloud environment.

Therefore, we need to set a scheduling algorithm with low complexity and resilience to performance variance. Kim [6] suggested workflow management scheme which schedule subtasks onto resources according to given budget or completion time. After investigating load of tasks in a workflow to decide QoS constraints distribution rate, the workflow is processed by designated scheme. As they belong to dynamic scheduling, each scheme effectively copes with performance variance problem. Also, contrary to Sakellariou's work [5], it doesn't require reassignment process, therefore

this heuristic instantly finds aimed schedule. However, Kim [6]'s workflow scheduling scheme is providing limited service range whose QoS is guaranteed.

In order to resolve the problem mentioned above, we consider a policy-based workflow scheduling scheme to efficiently expand QoS-guaranteed range. In this paper, we suggest QoS con-strained workflow scheduling scheme which utilizes task division policy. Also, we evaluate that how well our proposed scheme handle the performance variance.

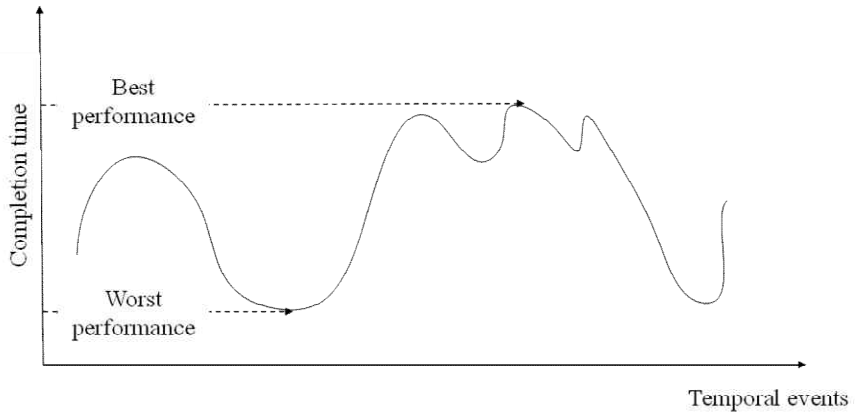


Fig. 1. Performance variance problem in generic computing environment [13]

3 Workflow Management System Model

3.1 A Layered Cloud Workflow Management System

We depict the cloud workflow management system as consisting of three main components. **Fig. 2** shows layered architecture of the proposed workflow management system. It has three core components – Workflow Scheduling Engine, Resource Provisioning Manager, and Policy Manager.

Workflow Scheduling Engine is in charge of interacting with Workflow Modelling Interface, which is the entrance for an application service. A user generates the processing request of scientific applications then submit the request to the management system with additional QoS related components using the Workflow Modelling Interface. Also, Workflow Scheduling Engine manages and executes those submitted workflows. Topology Analyzer module parses submitted workflow then interprets the topology of the workflow to figure out whether user-specified QoS constraints are enough to process the workflow request. Then, Policy Adaptor module communicates with Policy Manager to make workflow scheduling adaptively with user-specified QoS constraints. If the QoS constraints are sufficient to process the request, then Workflow Executor module initiates workflow scheduling and task execution. If they are not, then the request is rejected. Policy Manager maintains and decides workflow scheduling policies which are strategies to satisfy QoS constraints. Policy Manager chooses and provides optimal workflow scheduling policies to Workflow Scheduling Engine based on the workflow topology analysis. The policy contains scheduling strategies, such as deciding Virtual Machine (VM) resource service type which is mapped for each task, deciding the environment for task execution. Policy Decision Maker module decides the workflow scheduling policies

by referring to the Execution history repository and Policy repository. Decided policy is passed to Workflow Scheduling Engine to perform workflow scheduling according to the policy.

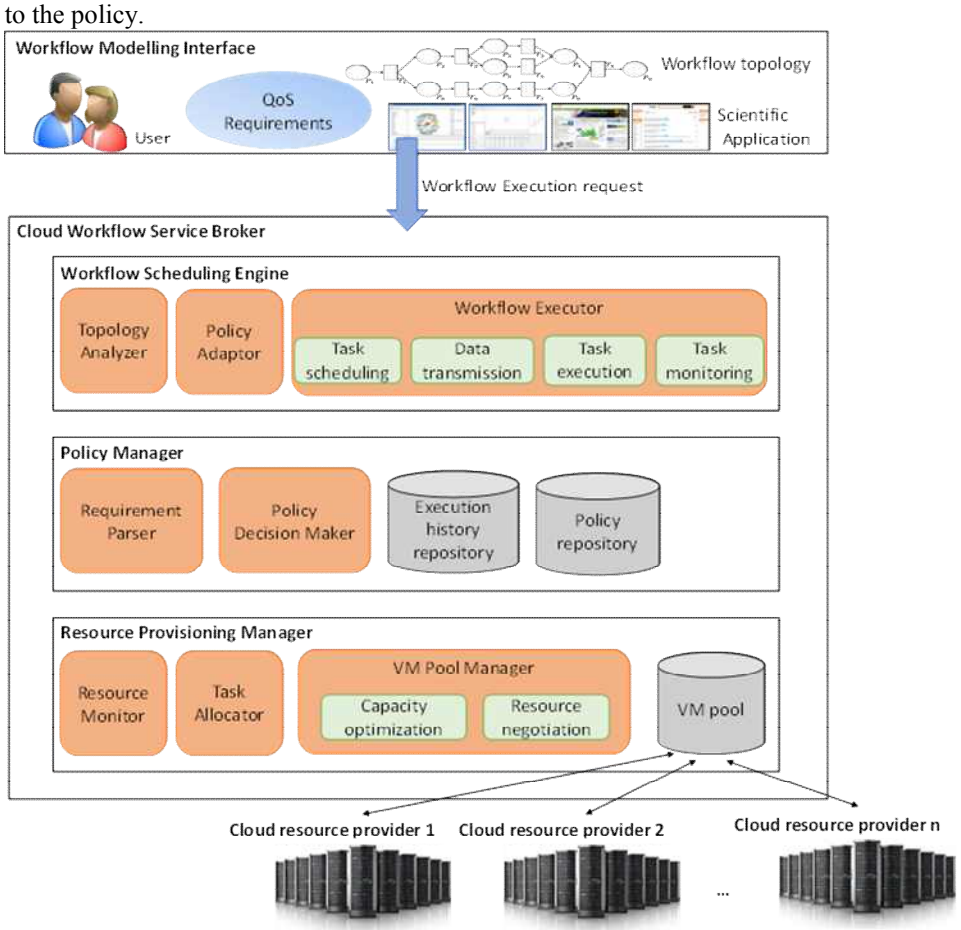


Fig. 2. A layered architecture of the cloud workflow management system

3.2 Workflow Management Description Using Petri-Net

Using Petri-Net, we describe workflow scheduling problem as follows.

Definition 1. Workflow We represent workflow W as $W = (P, T, A, M_0)$. $P = \{p_1, p_2, \ldots, p_k\}$ indicates a set of places which is used to represent the possible status of the workflow topology. $T = \{t_1, t_2, \ldots, t_k\}$ points out a set of transitions which is employed to depict the characteristics of each task. $A = \{(p_i, t_j), (p_j, t_i)\}$ is the connection between transitions and places.

Definition 2. Workflow scheduling problem with deadline constraint A problem which finds a schedule for a workflow $W=(P, T, A, M_0)$ to be executed within user-specified deadline *D* is defined as workflow scheduling problem with deadline constraint.

Definition 3. VM type A Cloud Virtual Machine Type is illustrated as $VT_j = [VT_{c_j}, VT_{hz_j}, VT_{m_j}]$ $(1 \leq j \leq m)$. Also, it is assumed that VM type set exists

as a finite set $VT = \{VT_1, VT_2, \ldots, VT_m\}$. In addition, rental price per unit time for arbitrary virtual machine VT_j is defined as C_{VT_j}.

Definition 4. **Application profiling matrix** Application profiling is the method to figure out average execution time for a task t_i when it is processed on virtual machine type VT_j and manage the execution time data in the form of table. The table is marked as application profiling matrix ***AP***. Each element AP_j^k which equals to $Ti_{VT_j}^{tt_k}$ is acquired from repeated execution.

The problem of making decision on mapping performance of different computing resources onto a job becomes the problem of selecting proper space from ***AP***. We define cost model to figure out workflow processing cost using cloud resources. We let rental cost per unit time for VM type C_{VT_j}. In addition, we denote VM usage time for the VM type VT_j as $Ti_{VT_j}^{tt_k}$. Then, total cost required to process given workflow is described as Eq. (1).

$$total\ cost = \sum_i C_{VT_j} \times Ti_{VT_j}^{tt_k} \tag{1}$$

4 A Proposal of QoS Constrained Workflow Scheduling Scheme Using the Colored Petri-Net Model

In order to overcome the problem that Kim [6]'s workflow scheduling scheme has, we consider applying task division policy in workflow scheduling. We define task division policy using the concept of arbitrarily divisible task [7]. In this paper, we only consider half-way task division not to make heuristic too complicated.

We define cost model to maximize profit while considering the cost and processing time. The profit Model for the scheduler is given to Eq. (2).

$$P_t = B - C_l - C_p \tag{2}$$

In the formula above, P_t indicates Profit. B is budget which is supplied by user. C_p is penalty cost. Additionally, C_l is cost for leasing VM(s) from cloud provider which is obtained as Eq. (3).

$$C_l = \sum_i^n C_{vm_i} = \sum_i^n (ut_{vm_i} + init_{flavor(vm_i)}) \cdot c_{flavor(vm_i)} \tag{3}$$

Basically, C_l is settled in the way of multiplying VM renal fee per unit time and duration of VM usage. In Eq. (3), $c_{flavor(vm_i)}$ means VM rental fee per unit time by different VM types. u_t denotes duration of VM usage for the task execution. Also, $init_{flavor(vm_i)}$ means VM initiation time.

Penalty cost C_p, which is caused by SLA violation, is obtained in Eq. (4) and Eq. (5).

$$C_p = \begin{cases} \alpha + \beta \cdot SV, & if\ SV^{upper} \geq SV > 0 \\ 0, & otherwise \end{cases} \tag{4}$$

$$SV = \begin{cases} CT - D, & if\ CT - D > 0 \\ 0, & otherwise \end{cases} \quad (5)$$

In Eq. (4) and Eq. (5), variable *SV* indicates the degree of SLA violation.

Workflow scheduling step represented by the colored Petri-Net

Step 1. Calculate the earliest completion time *CT* and load rate r(t_i) for each task

We investigate the earliest completion time *CT* and load rate r(t_i) for each task by transferring scheduling token from the last place p_{last} to the entrance place p_{ent}. Let ct_j be the earliest completion time of task j on the critical path [9], then *CT* is determined as shown in Eq. (6).

$$CT = \sum ct_j \quad (6)$$

Also, we can derive r (t_i) as Eq. (7).

$$r\ (t_i) = \frac{ct_i}{ct_1 + \sum_{i+1} ct_i} \quad (7)$$

Step 2. Find proper VM resource according to ***AP*** and *SV*

We transfer execution token from p_{ent} to p_{last} to find proper VM resources for task execution according to ***AP*** and *SV*. As we move execution token m, we multiply load rate r(t_i) with remaining execution time *RET(p)* to calculate the estimated subdeadline (*ESD*) for the task t_i. Also, we define SV Estimation Token m' in order to calculate penalty cost by proceeding them. According to *ESD*, we choose the cheapest resources when *SV* equals zero. When *SV* is greater than zero, we apply task division policy. Then, the task is divided until *SV* becomes zero. Or task division stops when the divided task becomes the task of fundamental size. Then, we drive load rate $r(t_i')$ for divided task to multiply it with *RET(p)*. Afterwards, according to *ESD* with task division policy applied, we choose the cheapest resources to process tasks.

4 Experiments and Evaluation

4.1 Experimental Environment

Fig.3 shows the structure of the experimental environment which consists of workflow designer, MySQL database, cloud broker, and OpenStack Cloud. We used MapChem [8] application to compose the services into the workflow topologies. MapChem [8] is an integrated application for collaborative pharmaceutical research. Each MapChem service includes QSAR data which needs to be processed. QSAR data, which includes the information for the chemicals, is denoted as .sdf file. We took files with different number of chemicals into the experiment, which are 25, 50, 100, and 200 respectively.

We performed the experiment with different workflow topologies, which is generated in workflow designer. Those topologies are different in number of tasks from 30, 50, and 70. Also, the shapes of topology are serial (*wt1*) [10], parallel (*wt2*) [11], and hybrid of *wt1* and *wt2 (wt3)* [3]. Once workflow topologies are generated in Workflow Designer, they are stored in MySQL Database as .xml file. We made workflow execution requests by specifying Pipeline ID of a workflow topology and deadline to evaluate how much the proposed algorithm expands QoS-guaranteed range and deals with resource performance variance compared to Kim [6]'s algorithm.

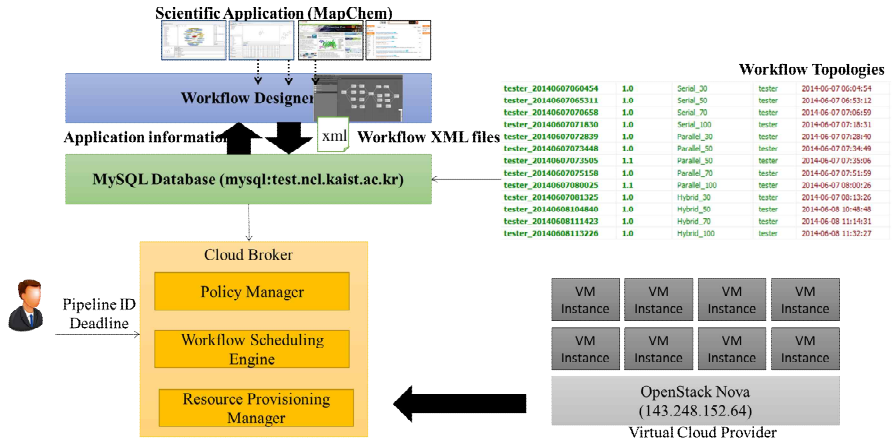


Fig.3. The structure of experimental environment

4.2 Experimental Results and Discussion

In order to evaluate the performance of the algorithm, we measured and compared the cost and completion time which is required to process the sample workflow set with different workload applying proposed workflow scheduling scheme, Phased Workflow scheduling Scheme with Division Policy (PWSD), and Kim [6]'s workflow scheduling scheme, Phase Workflow scheduling Scheme without Division Policy (PWS), respectively. Cost is the total expense needed to process each workflow. We put a price for each VM type by referencing the price model of GoGrid [12]: 0.06/second for m1.small, 0.09/second for m1.small_var, 0.12/second for m1.medium, 0.18/second for m1.btwml, and 0.24/second for m1.large. Completion time is time span taken for processing of each workflow topology.

In order to observe how much the proposed algorithm expands QoS-guaranteed range compared to PWS, we sent workflow execution request for each workflow topology while increasing user-specified deadline by 20 seconds. Also, in order to observe the robustness to resource performance variance of each scheme, we sent workflow processing request at the condition that deadline is fixed while task execution time delay increases proportionally by 0.05.

Fig.5. shows the actual completion time and cost for *wt1* with 50 tasks versus increment of deadline on PWSD and PWS. We are listing here only the graph for *wt1* with 50 tasks as graphs for *wt2* and *wt3* shows similar tendencies that *wt1* with 50 tasks displays. In this figure, dotted line indicates user-specified deadline. We can see that PWSD line is following the user-specified deadline, whereas PWS lines excesses dotted lines when user-specified deadline is too low. The low-deadline required regions in the graph is not able to be reached by PWS because even though PWS allocates the large type VMs to all the tasks then makes the fastest scheduling, it is still bigger than user-required deadline. For the region PWS can guarantee QoS, the lines of PWS and PWSD should be overlapped as no task division will be applied for PWSD, which will result in the same cost with the case when PWS is applied. Also, as shown in **Fig.6**, we can see that the proposed scheme shows better performance in

adapting to the delay time and completing the workflow execution on time than PWS as the proposed scheme produces QoS-satisfactory scheduling, or at least, makes only 2% execution time violation to the required deadline in the worst case.

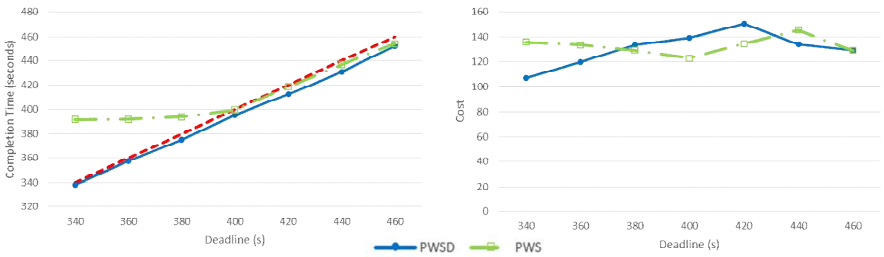
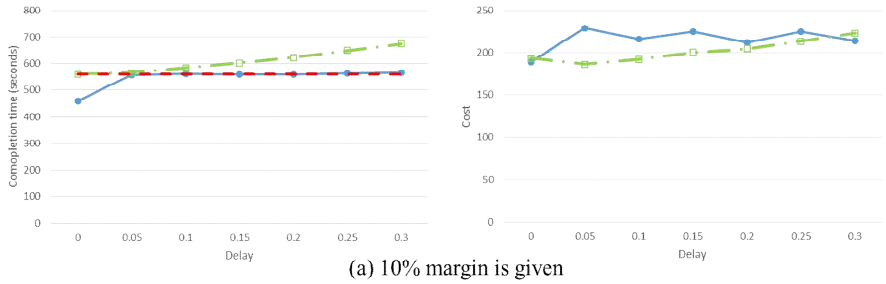
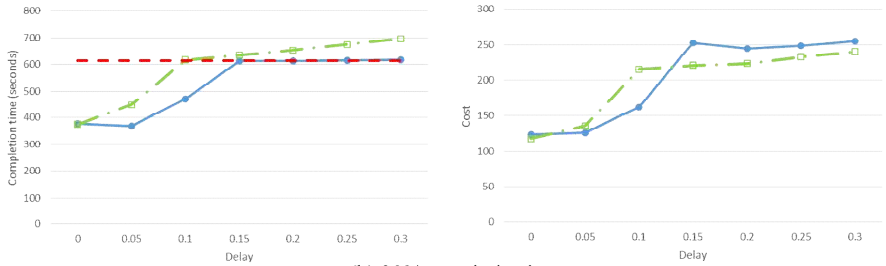


Fig.5. Performance comparison over cost and completion time while increasing deadline between the proposed algorithm and PWS for *wt1* with 50 tasks



(a) 10% margin is given



(b) 20% margin is given

PWSD PWS

Fig.6. Performance comparison over cost and completion time while increasing task execution time delay between the proposed algorithm and PWS for *wt1* with 70 tasks

5 Conclusion

In this paper, we proposed the policy-based QoS constrained workflow scheduling scheme which is a kind of the phased scheduling scheme. It has merits in dealing with the uncertainty of task execution time which is changed by the state of the VM resource and finding the near-optimal schedule for processing the workflow execution request without path guessing. Also, we suggested to apply the task division policy that divides then execute a task when SLA violation cost is big in order to expand QoS-guaranteed region and to improve robustness to resource performance variance.

In order to evaluate the performance, we measured the cost and the completion time for the proposed scheme and Kim [6]'s algorithm while increasing deadline or task execution delay. We could see that the proposed algorithm provides broader

QoS-guaranteed region and improved robustness to the resource performance variance. Therefore, we concluded that the proposed scheme surpasses PWS at providing broad QoS-guaranteed service region and handling the resource performance variance. In the future, we can develop our idea to consider and cope with other uncertainties, for example, dynamic VM resource price. Furthermore, we can make advance for the proposed scheduling scheme to consider multi QoS constraints simultaneously by determining new workflow policy.

Acknowledgments. This research was funded by the MSIP (Ministry of Science, ICT & Future Planning), Korea in the ICT R&D Program 2014 and supported by the MSIP (Ministry of Science, ICT&Future Planning), Korea, under the ITRC (Information Technology Research Center) support program (NIPA-2014(H0301-14-1020)) supervised by the NIPA (National IT Industry Promotion Agency

References

1. Blake, M. Brian, and David J. Cummings. "Workflow composition of service level agreements." Services Computing, 2007. SCC 2007. IEEE International Conference on. IEEE, 2007.
2. Wu, Linlin, Saurabh Kumar Garg, and Rajkumar Buyya. "SLA-based admission control for a Soft-ware-as-a-Service provider in Cloud computing environments." Journal of Computer and System Sciences 78.5 (2012): 1280-1299.
3. Yu, Jia, Rajkumar Buyya, and Chen Khong Tham. "Cost-based scheduling of scientific workflow applications on utility grids." e-Science and Grid Computing, 2005. First International Conference on. IEEE, 2005
4. Topcuoglu, Haluk, Salim Hariri, and Min-you Wu. "Performance-effective and low-complexity task scheduling for heterogeneous computing." Parallel and Distributed Systems, IEEE Transactions on 13.3 (2002): 260-274.
5. Sakellariou, Rizos, et al. "Scheduling workflows with budget constraints." Integrated Research in GRID Computing. Springer US, 2007. 189-202.
6. Kim, Daesun (2014). “Adaptive Workflow Scheduling Scheme Based on the Colored Petri-Net Model in Cloud.” Master’s thesis, Korea Advanced Institute of Science of Technology
7. Bharadwaj, Veeravalli, Debasish Ghose, and Thomas G. Robertazzi. "Divisible load theory: A new paradigm for load scheduling in distributed systems." Cluster Computing 6.1 (2003): 7-17.
8. PharosDreams. Available: http://www.pharosdreams.com/desktop/desktopsolution/home.html
9. Kelley Jr, James E. "Critical-path planning and scheduling: Mathematical basis." Operations Research 9.3 (1961): 296-320.
10. Sun, Sherry X., Qingtian Zeng, and Huaiqing Wang. "Process-mining-based workflow model fragmentation for distributed execution." Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on 41.2 (2011): 294-310.
11. Mao, Ming, and Marty Humphrey. "Auto-scaling to minimize cost and meet application deadlines in cloud workflows." Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis. ACM, 2011
12. GoGrid. Available: http://www.gogrid.com/
13. Kim, Byungsang (2013). “A Study on Cost Adaptive Cloud Resource Broker System for Bio-Workflow Computing.” Ph. D. Dissertation, Korea Advanced Institute of Science of Technology