A Phased Workflow Scheduling Scheme with Task Division Policy in Cloud Broker

Seong-Hwan Kim(✉), Kyung-No Joo, Yun-Gi Ha, Gyu-Beom Choi, and Chan-Hyun Youn

Department of Electrical Engineering, KAIST, Daejeon, Korea
{s.h_kim, eu8l98, milmgas, mosfetlkg, chyoun}@kaist.ac.kr

Abstract. In the science area, workflow management systems (WMS) coordinate collaborative tasks between researchers of many research organizations. Also, WMS effectively compose the high performance computing system with globally distributed computing resources. In addition, with the maturity of cloud computing technology, many researches try to enhancing the economic feasibility and system tolerability. While executing a workflow application, a workflow scheduler, which is in WMS, should recognize the dynamic status of resources and decide to assign appropriate resource on each task. With the negotiation procedure, users can ask for saving processing cost or shortening completion time. However, satisfying these multiple objectives at the same time is hard to achieve. Therefore, the existing workflow scheduling schemes try to find the near optimal solution with heuristic approaches. In this paper, we propose heuristic workflow scheduling scheme with petri-net workflow modeling, resource type mapping in accordance to workload ratio and policy based task division to guarantee the deadline constraint with minimum budget consumption.

Keywords: Workflow scheduling · Colored patri-net · Task division policy · Cloud computing

1 Introduction

In an aspect of modularization, automated processing, expandability, collaborative works and ease of control and monitor, workflow is appropriate tool for modeling complicated application. Especially, a scientific workflow is the computerized automation of a scientific process, in whole or part, which usually streamlines a collection of scientific tasks with data channels [1]. Since the correct integration of these distributed services may require an efficient management scheme and tools, a well-designed workflow management system in cloud is required to completely define, manage, monitor, and execute scientific workflows through the execution of tasks whose execution order is driven by a computerized representation of the workflow logic. To process request of computing workflow from user, workflow scheduling (resource planning), which allocate available computing resources to each workflow tasks with amount and allocation time of resources, is required. However, it is difficult to make optimal scheduling with consideration of multiple QoS (Quality of Service),

V.C.M. Leung et al. (Eds.): CloudComp 2014, LNICST 142, pp. 76–86, 2015.
DOI: 10.1007/978-3-319-16050-4_7

inter-task dependency and dynamic status of resources. In cloud environment, we should also consider the variation of resource capacity over time, the heterogeneity of resources and the VM (Virtual Machine) leasing cost model from cloud service providers. Because it is generally considered as NP-complete problem to solve multi object-workflow scheduling problem, many previous studies generally utilize heuristic strategies to acquire real-time decision with relatively high optimality [2].

Yu [3] proposed a workflow management system in grid with MDP (Markov Decision Process) based workflow scheduling scheme while guaranteeing the assigned deadline. By using the workflow partitioning strategy, they try to find optimal solutions (assign sub-deadline) on each partial task groups with MDP method. To achieve deadline guarantee, they consider critical path concept. By assembling local optimums on each partitioned task groups, they tried to find the global near optimum. However, since their algorithm can't consider complexity of cloud price policy, cost can be wasted in cloud environment. Although, MDP is calculated on partial task groups, MDP is relatively complex to compute.

In this paper, we propose heuristic workflow scheduling scheme with petri-net workflow modeling, resource type mapping in accordance to workload ratio and policy based task division by resolving problems in Phased Workflow Scheduling Scheme [4] to guarantee the deadline constraint with minimum budget consumption.

2 Workflow Scheduling in a Cloud Broker

In this paper, we consider Users, Cloud Brokers and Cloud Service Providers (CSP) as actor of our system framework. When user composite and request their own workflow instance to cloud broker with QoS constraints and budget contraction, it should be scheduled by Workflow Scheduling Engine (WSE) in cloud broker. Then, WSE should execute each task depending on scheduling decision in available cloud resources which are provided by CSP. With cooperation of Resource Provisioning Manager (RPM) which give abstraction to the VM leasing contraction, CSP cost policy and physical details of VMs, WSE can easily allocate tasks into proper VMs. RPM manages the Virtual Machine Pool (VMP) in cloud broker to provide resources into multiple application sets in efficient way. Because RPM makes reasonable contraction between multiple CSPs with the profiling and comparing manners, it is more efficient then 1:1 VM leasing contraction between applications and CSPs. Therefore, with the scheduling to find optimal resource planning to execute workflow and resource provisioning to make efficient proxy contraction, user doesn't need to consider the details of complex procedure to execute workflow after making contraction with cloud broker. Because billing contraction also contains service level violation penalty which is pay back cost from cloud broker when cloud broker can't satisfy service level, cloud broker should reject the contract when it is unable to achieve. There are many ways to set the violation penalty cost, but we will use linear penalty cost model [5] in this paper.

Figure 1 shows functional architecture of the proposed cloud broker. It simply has two core components – Workflow Scheduling Engine and Resource Provisioning Manager.

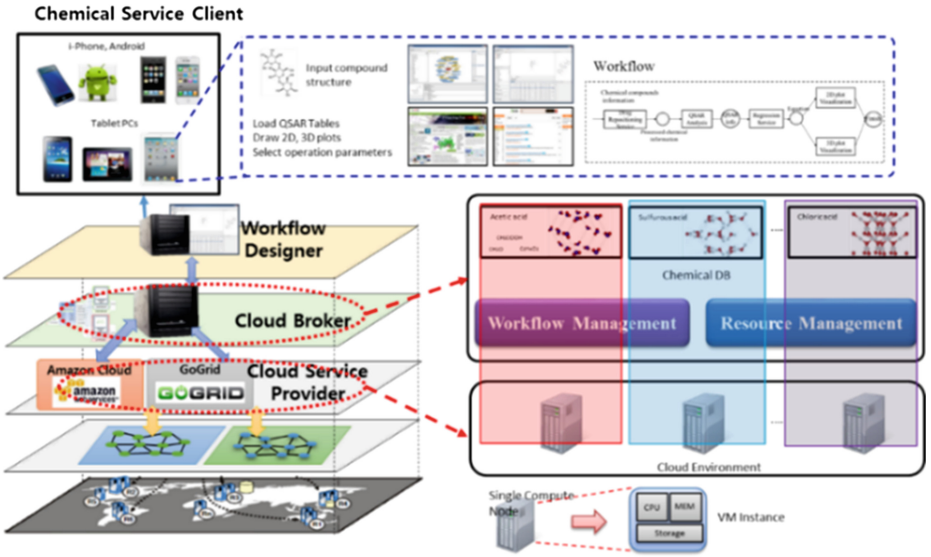


Fig. 1. Architectural model of the proposed cloud broker

Workflow Scheduling Engine (WSE). WSE provides unified user interfaces for creating, managing and executing workflow application service. User can compose the workflow instances with deployed services which are registered by participants formerly with service profiling data. When execution requests are given, the WSE should decide to assign appropriate resource on each task based on the scheduling scheme. Finally, with task dependencies, collaboration with RPM and resource mapping table, WSE can execute each tasks to the available VMP resources in order.

Resource Provisioning Manager (RPM). RPM manages virtual machine pool (VMP) as a logical container. The resources in the VMP represent the leased VMs from the underlying CSPs. Therefore, the RPM allocates or deallocates the resources in VMP based on its own resource leasing strategy. Also, RPM shares resources among various applications.

A problem which finds a schedule for a workflow $W = (P, Tr, A)$ to be executed within user-specified deadline D is defined as Workflow scheduling problem with deadline constraint. That is, deciding assigned computing resources-to-be set $R = \{R_1, R_2, \ldots, R_n\}$ and assigned time set $T = \{T_1, T_2, \ldots, T_{\mathrm{n}}\}$ according to petri-net model is illustrated as the problem.

Cloud Virtual Machine Type is defined by combination of parameters which are time-invariant and continuously capable of being guaranteed by cloud service providers. (e.g. number of CPU cores: VT_c, clock rate of a CPU: VT_{hz}, memory size of the virtual machine: VT_m, storage size of the virtual machine: VT_s) In this paper, we only consider factors which are directly related with job processing time. Therefore, a Cloud Virtual Machine Type is illustrated as $VT_j = [VT_{c_j}, VT_{hz_j}, VT_{m_j}]$ and exists as a finite set $VT = \{VT_1, VT_2, \ldots, VT_m\}$. In addition, leasing cost per unit time for arbitrary virtual machine VT_j is defined as C_{VT_j}.

Application Profiling is the method to figure out expected 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 row of table represents different task type $tt_k (= t_{i,type}, 1 \leq k \leq l)$ and the column represents virtual machine type VT_j Each execution time data $T_{VT_j}^{tt_k}$ is the average value acquired from enough times of repeated execution.

Also, we define cost model to figure out workflow processing cost using cloud resources. When denote VM usage time for the VM type VT_j as $T_{VT_j}^{tt_k}$, Then, execution cost required to process given workflow is described as following equation

$$execution\ cost = \sum\nolimits_i \mathrm{C}_{VT_j} \times T_{VT_j}^{tt_k}. \tag{1}$$

3 Phased Workflow Scheduling Scheme with Task Division Policy

In order to resolve the cost minimization while deadline-guaranteed workflow scheduling problem, we should pay the least at individual task processing by assigning appropriate resource. Also, the whole workflow schedule made by individual task scheduling should satisfy user-specified deadline. Kim [4] proposed phased workflow scheduling scheme. With the colored petri-net model, colored token which conducts different action on workflow topologies according to its color is defined to control the scheduling of given workflow topology. The scheme is composed with two phases: First one is Scheduling Phase, which decides the workload proportion and allocate sub-deadline into each task by backward token forwarding of scheduling token through petri-net. Second one is Execution Phase, assigns proper resource according to the load proportion of each task not to violate sub-deadline by forwarding execution token. As a result, the path which is in charge of the most portion of load in the workflow, namely, critical path is found out and delivered by the scheduling token. Because scheme use quietly simple and intuitive heuristic, it is fast enough to utilize in real-time workflow management system. Although this scheme takes advantage of static estimated task processing time, it is categorized as the dynamic scheduling as it carries out task scheduling according to the remaining service level indicator. Therefore, workflow management system can endure system fault from resources. However, when deadline which is shorter than the shortest estimated execution time of given workflow is required from user, it is not able to find a schedule which can satisfy user-specified deadline because of its finite resource set. Also, with the finite resource set, if processing delay exceed certain level, we can't guarantee the deadline, because we can't utilize better resources to compensate delays for processing remaining tasks.

To overcome the problems, we defined divisible task using the concept of arbitrarily divisible task. A task t_i which can be partitioned as they do not have any precedence relations and all elements in the task t_i are identical type of processing is defined as divisible task [6]. A divisible task t_i whose critical degree (cd), which is defined as the maximum number of partition of a task, is n can be divided into sub-tasks $\left[t_{i,1}, t_{i,2}, \ldots, t_{i,n}\right]$. For these divisible task, we can reduce job processing time by

distributing partitions of tasks to multiple resources. Though task division makes workflow processing time reduced, task division is not always carried out as it is not appropriate to reduce required cost and processing time, namely, the goal of workflow scheduling problem. Therefore, we should determine when we have to proceed division based on processing time and required cost.

In this paper, we only consider half-load division to control scheduling complexity and size of historical data in low level. In addition, it is assumed that the load of each partitioned task is the same. Also, subtask type is defined by its size of load. r partial tasks gathers to compose subtask type $tt_{k,\{r\}}$ whose $cd(tt_{k,\{r\}}) = r$. Additional application profiling for all kinds of task bunches is required to measure execution time of each subtask type $tt_{k,\{r\}}$ on different resource type VT_j. Example of application profiling table is shown in Table 1.

Table 1. Example of application profiling for divisible task whose *cd* equals 8

	$tt_{k,\{8\}}$	$tt_{k,\{4\}}$	$tt_{k,\{2\}}$	$tt_{k,\{1\}}$
VT_1	$AP_1^{k,\{8\}} = T_{VT_1}^{tt_{k,\{8\}}}$	...	...	$AP_1^{k,\{1\}} = T_{VT_1}^{tt_{k,\{1\}}}$
VT_2	...	...	...	...
VT_3	$AP_3^{k,\{8\}} = T_{VT_3}^{tt_{k,\{8\}}}$	...	...	$AP_3^{k,\{1\}} = T_{VT_3}^{tt_{k,\{1\}}}$

Workflow Scheduling Phases

When a user submits a workflow execution request, then the cloud broker should map proper resources to tasks. We introduce the QoS constraints scheduling scheme which is used to schedule each task onto appropriate VM type. We assume that token forward and backward matrix is already extracted by analysis of workflow topology. Then we can move our token with multiplication of token status vector and token forward/backward matrix.

Phase I. Calculate the load rate r(p) for each task

We set the initial marking of token vector for first phase as $M = [0...01]$. Then the token moves through backward matrix along the workflow topology path reversely investigating each task's the load rate. The load rate $r(p)$ on a place p is the rate of the transition (task)'s relative load compared to relative load of its critical path.

$$r(p) = \frac{rl(p)}{cpl(p)}. \tag{2}$$

Also, relative load is defined as average execution time for a task on VM types:

$$rl(p) = \mathrm{avr_j}\left(T_{VT_j}^{type(p^*)}\right) = \frac{1}{\mathrm{m}} \cdot \sum\nolimits_j T_{VT_j}^{type(p^*)}. \tag{3}$$

In addition, Critical path on a task is defined as set of following tasks which is composed of biggest relative load [7]. Then, we can figure out critical path load:

$$cpl(p) = f(x) = \begin{cases} \max_{p^{**}} cpl(p^{**}), & if\ p^{**}\ exist \\ 0, & otherwise \end{cases}. \tag{4}$$

Because we assume that we already collected the application profiling matrix for all tasks on different VM type, we can calculate load rate on entire workflow in static way.

Phase II. Allocate sub-deadline and assigns proper resource according to the load proportion of each task

We set the initial marking of token vector for second phase as $M = [10\ldots0]$. Then the token moves through forward matrix along the workflow topology path investigating each task's the sub-deadline and assigning cheapest VM which can guarantee sub-deadline. Sub-deadline is ways for guarantee the entire deadline. When we allocate sub-deadline properly, and if each task can guarantee the sub-deadline, we can achieve successful scheduling. Therefore, we allocate sub-deadline rationally based on remaining time (deadline D subtracted by current execution time $T_c(m)$) and load rate.

$$sd(p^*)\{= rl(p) \cdot (D - T_c(m)). \tag{5}$$

Then based on Application Profiling matrix, we can find cheapest VM which can execute task in sub-deadline, and can execute task into available VM with the support of Resource Provisioning Manager. When execution is over, token can be moved to next step.

When there are no available resource types to guarantee sub-deadline, it may cause deadline violation for entire workflow. Therefore, we token is not moved to next step and should apply task division policy in phase III.

Phase III. With the cost model check whether task division policy is necessary

As mentioned before, we should determine when we have to proceed division based on processing time and required cost. Also, we should consider the service level violation penalty on required cost. To determine profit in one dimension we define cost model to maximize profit while considering the cost and processing time. The profit Model for the scheduler is represented as follows:

$$P_t = B - C_l - C_p \tag{6}$$

In the formula above, P_t indicates Profit. B is budget which is supplied by user. C_l is cost for leasing VM(s) from cloud provider which is specified in (1). Penalty cost C_p, which is caused by SLA violation, is represented as follows:

$$C_p = \begin{cases} \alpha + \beta \cdot SV, & if\ SV > 0 \\ 0, & otherwise \end{cases} \tag{7}$$

$$SV = \begin{cases} ECT - D, & if\ ECT - D > 0 \\ 0, & otherwise \end{cases} \tag{8}$$

In Eqs. (7) and (8), variable SV indicates the degree of SLA violation. There are many models of violation penalty cost, but in this paper we use simple linear violation

penalty model (7) [5]. As shown in (8), SV is described as subtraction deadline *D* from estimated completion time ECT.

We estimate SLA violation SV and SV' in a deterministic way. We define SV Estimation Token m_e in order to calculate penalty cost by proceeding them. Initially, the location of m_e is replicated from current execution token m. Also, each token save temporal current execution time $T_{tc}(m_e)$ which is replicated from current execution time $T_c(m)$. Then, by moving each token with forward matrix, calculating the estimated-sub-deadline, allocate temporal VM, cumulating the $T_{tc}(m_e)$ for each transition with the method in phase II until token reach the final place, we can get the estimated execution time $T_{tc}(m_e)$ in heuristic way.

Table 2. Algorithm of task division policy in workflow scheduler

Algorithm. Division Policy

Input: *workflow topology tp, workflow topology with half division tp', current execution token matrix m, current execution time* $T_C(m)$, *workflow deadline D, budget B, token forward matrix F*

Output: *Decision of division*

Copy execution token set m onto SV estimation token set m_e and copy current execution time $T_c(m)$ onto temporal current execution time $T_{tc}(m_e)$.

for workflow topology *{tp, tp'}*

while m_e is not equal to [0 … 0 1]

Let P= $\{p_1, p_2, \ldots, p_n\}$ be places which has SV estimation tokens at current stage i and estimation token set at stage i as m_e^i

for p = each element in P

$sd(p^*) = rl(p) \cdot T_{tc}(D - m_e^i)$

$p^*.temporal\_vm = VT_{\left\{j \middle| T_{VM_j}^{type(p^*)} < sd(p^*), \min\left(C_{VT_j}\right)\right\}}$

Let $Tr = \{t_1, t_2, \ldots, t_k\} =^* p$

$T_{tc}(m_e^i) = T_{tc}(m_e^{i-1}) + \max_j \left(T_{t_j.temporal\_vm}^{type(t_j)}\right)$

endfor

$m_e^{i+1} = F \cdot m_e^i$ //token forwarding

endwhile

if $T_{tc}(m_e) - D > 0$

$C_p = \alpha + \beta \cdot SV = \alpha + \beta \cdot (T_{tc}(m_e) - D)$

else

$C_p = 0$

endif

$C_l = \sum_i C_{t_i.temporal\_vm} \times T_{t_i.temporal\_vm}^{type(t_i)}$

$P = B - C_l - C_p$

endfor

return $P_{tp} < P_{tp'}$ // If $P_{tp} < P_{tp'}$, return value is true. Otherwise return value is false

Then with the comparison of profit between the non-division case and division case (half division), we can determine the cost efficient decision. Therefore we should calculate profit P for non-division case and profit P' for division case. For calculating profit P' of division case, application profiling for divisible task (Table 1) will be needed.

If P < P', apply the half division and return to phase II. If division case not yet guarantee the deadline, division can be occurred recursively until task is no further divisible (critical degree equals 1). If P > P', allocate biggest VM to transition and return to phase II for forwarding to next step (Table 2).

4 Experimental Evaluation

4.1 Experiment Environment

The experimental environment consists of workflow designer, cloud broker, and OpenStack Cloud Infra as shown in Fig. 2. We composed three workflow topologies using workflow designer, and requested execution of one workflow within a certain deadline. The task request was sent to the cloud broker, and then a cloud broker did allocated each sub-task within the sub-deadline. The workflow management system used in the experiment was implemented for OpenStack cloud to run each sub-task.

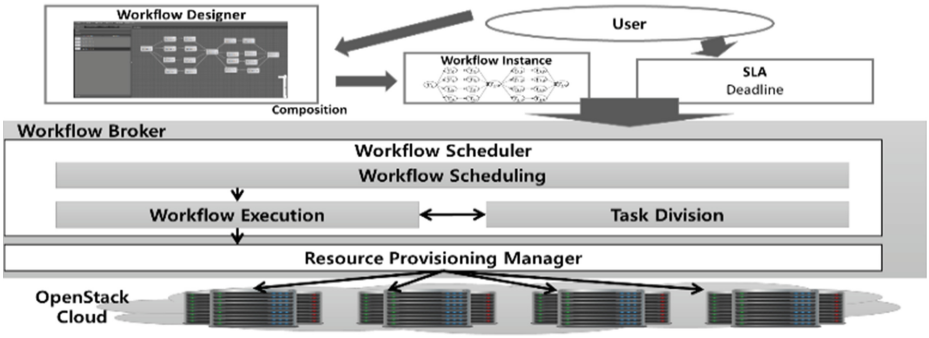
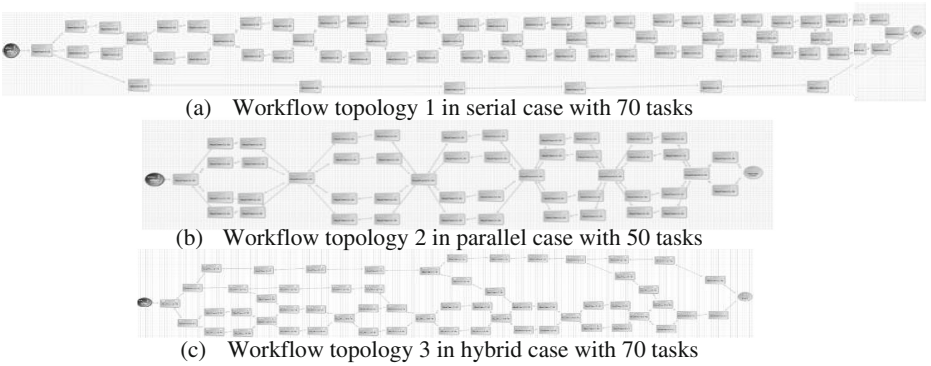


Fig. 2. Experimental environment

Three examples of workflow applications are shown in Fig. 3. Workflow type 1 (a) has some sequentially connected split-merge pairs that have some parallel tasks. Workflow type 2 (b) has little split-merge pairs. However, they have lots of parallel tasks. Workflow type 3 (c) are composed of hybrid structures. We used 4 kinds of task type in random distribution for each tasks.

4.2 Experimental Results and Discussion

We requested the execution of each workflow applications repeatedly with various SLAs (deadlines), and investigated the actual execution time and cost. Additionally, we



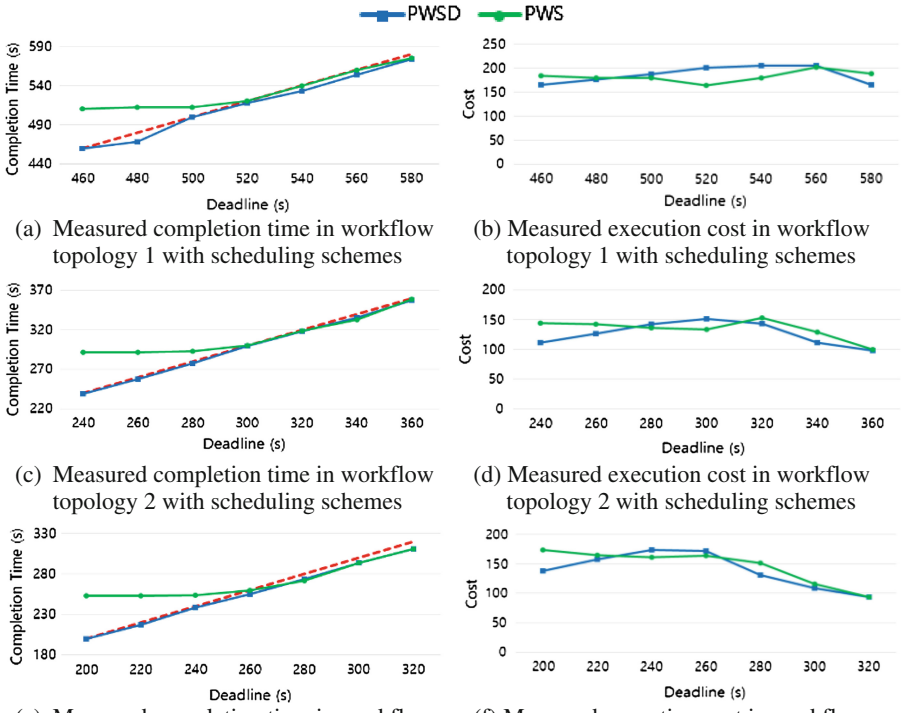
(a) Workflow topology 1 in serial case with 70 tasks

(b) Workflow topology 2 in parallel case with 50 tasks

(c) Workflow topology 3 in hybrid case with 70 tasks

Fig. 3. Three types of workflow topology used in experiment

calculated the difference between the given SLA and the actual execution time or cost to see how well each policy guaranteed the given SLA. Therefore, we should check whether the proposed scheme work properly so that the actual execution time follows



(a) Measured completion time in workflow topology 1 with scheduling schemes

(b) Measured execution cost in workflow topology 1 with scheduling schemes

(c) Measured completion time in workflow topology 2 with scheduling schemes

(d) Measured execution cost in workflow topology 2 with scheduling schemes

(e) Measured completion time in workflow topology 3 with scheduling schemes

(f) Measured execution cost in workflow topology 3 with scheduling schemes

Fig. 4. Experimental results with scheduling schemes (PWSD and PWS) on completion time and execution cost

the given deadline. Proposed scheme (PWSD) is compared with phased workflow scheduling without division policy (PWS) [4]. Because of the limitation on amount of cloud resources compared to the workload on each workflow instances, experiment was performed in emulation.

Figure 4 shows the experimental results according to the scheduling scheme. (a), (c), and (e) represent the actual execution time with different workflow types versus increment of deadline on PWSD and PWS scheme. Dotted lines is plotted to support comparison with the QoS (deadline) constraint. Line with circular points shows experimental result with PWS scheme and line with square points shows experimental result with PWSD scheme. We can find that three graphs show the similar result. When the deadline is too low, the deadline policy cannot meet the deadline in PWS although the broker allocates whole computing resource with large VM types. Therefore, there are some QoS violations in low deadline requirement. In this region, request should be rejected before it pass the admission controller. However, in case of PWSD, it guarantee the QoS region, which PWS can't guarantee, with task distribution and division policy. When the deadline is given adequately, the both scheme can schedule for adequate VM types with assurance of the requirement and might work in same way. In advance, (b), (d), and (f) show the execution costs of different workflow types when we change the deadline. In PWS scheme and, larger type VM are used frequently in order to meet deadline. Also because of violation penalty cost, the execution cost tends to be more expensive, when the deadline is small. However, in PWSD in the same manner, the execution cost does not increase because broker do not have to pay violation penalty. Because we do not consider the management load (split and merge) and data transmission delay between tasks, cost goes down with the profit from frequent task division.

In this experiments, we can conclude that the division policy schedules workflow with guaranteeing the deadline, while trying to use resources in efficient way. We checked this policy works well in various types of workflow.

5 Conclusion

We propose the adaptive workflow scheduling scheme based on the colored Petri-Net model which tries to schedule in the cheapest way while assuring the deadline. Our model uses the phased scheduling model. Therefore, it can schedule dynamically with low complexity. Also, we showed that our result ensures the deadline. Our work was to distribute sub-deadline to each tasks based on its importance compared to the rest of the workflow. We can extend this idea to distribute budgets to each tasks based on the same importance and make user to choose the policy they want. Further work will contain these extensions. Also, we can generalize the problem by using the utilization concept consisting of deadline and budget.

Acknowledgments. This work was supported by the ICT R&D program of MSIP/IITP [10038768, The Development of Supercomputing System for the Genome Analysis] and 'The Cross-Ministry Giga KOREA Project' of The Ministry of Science, ICT and Future Planning, Korea. [GK13P0100, Development of Tele-Experience Service SW Platform based on Giga Media].

References

1. Jeong, S., Jo, Y.M., et al.: A novel model for metabolic syndrome risk quantification based on areal similarity degree. IEEE Trans. Biomed. Engi. **61**(3), 665–679 (2014)
2. Ren, Y., Kim, S.-H., et al.: A cost-efficient job scheduling algorithm in cloud resource broker with scalable VM allocation scheme. KIPS Trans. Softw. Data Eng. **1**(3), 137–148 (2012)
3. Yu, J., Buyya, R., Tham, C.K.: Cost-based scheduling of scientific workflow applications on utility grids. In: First International Conference on e-Science and Grid Computing, pp. 8–147. IEEE (2005)
4. Kim, D.-S.: Adaptive workflow scheduling scheme based on the colored petri-net model in cloud. Master's thesis. KAIST, Daejeon, Korea (2014)
5. Wu, L., et al.: SLA-based resource allocation for software as a service provider (SaaS) in cloud computing environments. In: 11th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid), pp. 195–204. IEEE (2011)
6. Bharadwaj, V., et al.: Divisible load theory: a new paradigm for load scheduling in distributed systems. Cluster Comput. **6**(1), 7–17 (2003)
7. Kelley Jr., J.E.: Critical-path planning and scheduling: mathematical basis. Oper. Res. **9**(3), 296–320 (1961)