A QoS Constrained Workflow Scheduling Scheme Using Task Division in Cloud

Yun-Gi Ha, Seong-Hwan Kim, Ji-Soo Choi, Seong-Min Song, Chan-Hyun Youn
Department of Electrical Engineering, KAIST
E-mail{yungi.ha, s.h_kim, jisoochoi, songsm87, chyoun}@kaist.ac.kr

Abstract

Various scientific workflow applications take advantage of cloud infrastructures which enables them process data with efficiency. In the cloud computing environment, workflow scheduling is the core which impacts on the job processing performance, the resource utilization and the throughput of workflow management system. We propose a Quality of Service (QoS) constrained workflow scheduling scheme using the task division in order to assure delivery of workflow service within certain deadline and minimized cost constraint. We show the effectiveness of the proposed scheme by comparing the completion time and cost of the workflow processing request for the workflow scheduling scheme with and without task division, while increasing the deadline constraint.

1. Introduction

Cloud computing provides its customer computing resources with scalability in pay-as-you go manner. These resources are provisioned and made available easily, which drags many organizations to it.

Scientific workflows, which are used to model applications of complex large scale data analysis, also benefits from cloud infrastructures. These workflow applications require assurance for service level that each request finishes within certain time and cost constraint. Thus, workflow scheduling needs to analyze a user's Quality of Service (QoS) constraints then map each task onto the appropriate virtual machine so that the execution result is able to satisfy user requirements [1]. Also, it is not the goal for workflow scheduling to find the way to make execution at the earliest, but to make it under the user specified constraints.

[2] proposes a workflow scheduling algorithm which assigns executable tasks onto the least expensive computing resources. Executable task indicates unexecuted task of which the execution for parent tasks is finished. Once it finds a schedule, it compares the expected completion time to the deadline constraint. If the expected completion time is larger than the deadline constraint, it removes the least expensive computing resource from the resource pool than repeats whole procedure until it finds the suitable schedule. This algorithm has some limitations. It may incur many iterations to meet the deadline constraint. Also, as it is kind of the static scheduling scheme, the solution can be not optimal because situation is not always changeless [3].

[4] introduces the workflow scheduling scheme which partitions a workflow then distributes the time constraint to each task. On the contrary to [3], while taking care of workflow topology, it utilizes the subdeadline for each task and overall deadline to find the suitable resource. Hence, this algorithm produces decent plan for the executable task without repeating subdeadline decision. However, it doesn't consider one of the attributes in cloud that it supports a user to utilize considerable computing power in pay-as-you-go manner, when the tasks are partitionable and transferable their load.

Our involvement in this article is twofold. First, the proposed scheduling scheme investigates each task's fastest completion time along the workflow topology then decides its distribution rate. Then, it allocates VM onto the task according to the remaining deadline and the task's distribution rate. Second, when there is no match, it divides and executes task based on the task's minimum size and the task's subdeadline, which is called as divisibility factor [5].

2. System Architecture for the Cloud Workflow Management System

We depict our cloud workflow management system which consists of three main components. 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.

3. A Workflow Scheduling Scheme Using Task Division

Kim [6] proposed a workflow scheduling scheme using Petri-Net which has two phases, scheduling phase and execution phase respectively. It finds out the critical path [7] for the given workflow, which is the path that decides the deadline for the workflow, then assigns suitable VM according to the load proportion of each task.

We define a task t_i which can be partitioned into two subtasks which have the same load 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 [5]. Examples of divisible tasks are aln process of BWA applications [8] and chem computing of MapChem [9] which usually take form of loop and don't have dependency within a single task. Also, we define the divisibility factor $df\,(t_i)$ for each task to set the division threshold, which is the maximum number of partitioned subtask. In this paper, we only consider half division to avoid high complexity in task scheduling. Therefore, a task t_i of which the divisibility factor $df\,(t_i)$ equals to n (power of 2) is divided into two subtask $\{t_{i,1}, t_{i,2}, \dots, t_{i,n/2}\}$ and $\{t_{i,\frac{n}{2}+1}, t_{i,\frac{n}{2}+2}, \dots, t_{i,n}\}$.

We describe our proposed scheduling scheme step by step using Petri-Net. Each workflow processing request consists of workflow topology $\boldsymbol{W}$ and deadline D. Workflow topology $\boldsymbol{W}$ is represented as $\boldsymbol{W}= (\boldsymbol{P}, \boldsymbol{T}, \boldsymbol{A})$. $\boldsymbol{P} = \{p_1, p_2, \dots, p_m\}$ is a set of places which surrounds each tasks. $\boldsymbol{T} = \{t_1, t_2, \dots, t_k\}$ is a set of transitions which represents each task of the workflow topology. $\boldsymbol{A}$ is a set of arcs which is used to connect transition – place or place – transition pair.

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

We compare the earliest completion time with user specified deadline. We set the initial marking of token as $\boldsymbol{M_0} = [0\ 0\ 0\ \dots 0\ 1]$**.** Then the token moves along the workflow topology path reversely investigating each task's earliest completion time and the load rate of the each task.

The earliest completion time *CT* is the sum of the earliest completion time of task on the critical path [7]. We assume that we already collected the execution time information for all tasks on different VM type. Let α_j be the earliest completion time of task j on the critical path, then *CT* is determined as shown in Eq. (1).

$$CT = \sum \alpha_j \qquad (1)$$

Also, we can derive r (t_i) as Eq. (2).

$$r(t_i) = \frac{ct_i}{ct_i + \sum_{i+1} ct_l} \qquad (2)$$

Step 2**.** Divide the task according to the divisibility factor $df\,(t_i)$, the load rate $r(t_i)$ for each task and remaining deadline while considering the cost model then allocate proper resource

We set the allocated execution time for each task $aet\ (t_i)$ based on remaining deadline and the load rate $r(t_i)$.

$$aet\ (t_i) = r(t_i) \cdot (D - T_c(t_i)) \qquad (3)$$

In Eq. (3), $T_c(t_i)$ indicates total execution time spent before processing task t_i. We use task profiling matrix shown in **Table 1** to find the cheapest way to process the task within $aet\ (t_i)$.

When there exists a VM which can execute the task within $aet\ (t_i)$, then task division is not considered. On the contrary, there is no satisfactory VM, we divide the task half then look up task profiling matrix to find the way we can execute the task within $aet\ (t_i)$. If it still doesn't have the adequate VM, then we repeat division until we find the proper VM or the task meets the end of $df\,(t_i)$.

If the task division is applied, then we compare the profit calculated from the cost model in Eq. (4) for

Table 1. Example of task profiling matrix for task t_i of which $df(t_i)$ equals to 8

$df(t_i)$ / VM type	8	4	2	1
VT_1	$T^{VT_1}_{t_{i,\{8\}}}$	$T^{VT_1}_{t_{i,\{4\}}}$	$T^{VT_1}_{t_{i,\{2\}}}$	$T^{VT_1}_{t_{i,\{1\}}}$
VT_2	$T^{VT_2}_{t_{i,\{8\}}}$	$T^{VT_2}_{t_{i,\{4\}}}$	$T^{VT_2}_{t_{i,\{2\}}}$	$T^{VT_2}_{t_{i,\{1\}}}$
VT_3	$T^{VT_3}_{t_{i,\{8\}}}$	$T^{VT_3}_{t_{i,\{4\}}}$	$T^{VT_3}_{t_{i,\{2\}}}$	$T^{VT_3}_{t_{i,\{1\}}}$

division case and non-division case to proceed the workflow scheduling in more profitable way.

$$P_{t_i} = B - C_l - C_p \qquad (4)$$

In Eq. (4), P_{t_i} indicates the profit until the scheduler processes task t_i. C_l is total VM leasing cost. C_p is penalty cost which is caused by SLA violation.

$$C_p = \begin{cases} \alpha + \beta \cdot SV, & \text{if } SV > 0 \\ 0, & \text{otherwise} \end{cases} \qquad (5)$$

$$SV = \begin{cases} T^{VT_j}_{t_{i,\{k\}}} - aet(t_i), & \text{if } aet(t_i) - D > 0 \\ 0, & \text{otherwise} \end{cases} \qquad (6)$$

In Eq. (5) and 6, SV indicates SLA violation, which is caused by mapping the task onto the VM which cannot process the task within $aet(t_i)$.

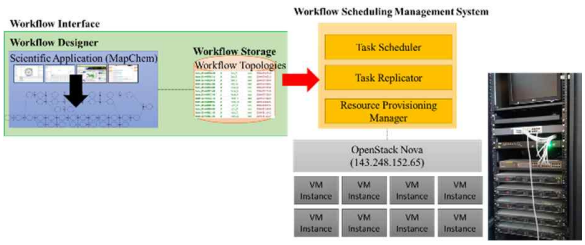
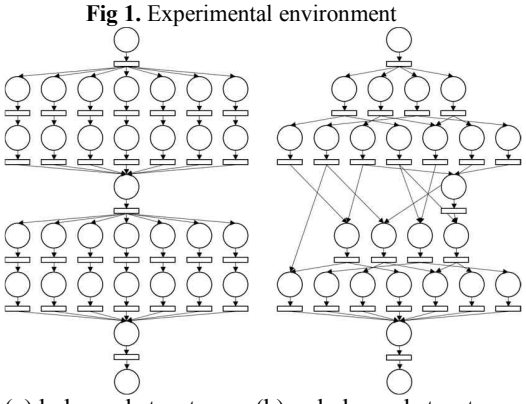


Fig 1. Experimental environment



(a) balanced structure (b) unbalanced structure

Fig. 2. Workflow topologies for the experiment [1]

4. Evaluation

4.1 Experiment Setting

Fig.1 shows the structure of the experimental environment which consists of workflow designer, MySQL database, cloud broker, and OpenStack Cloud. We used MapChem [9] application to compose the services into the workflow topologies.

We performed the experiment with workflow topologies with balanced structure and unbalanced structure [1] as shown in **Fig.2**. Within these workflows, tasks are randomly generated. We made workflow execution requests by specifying Pipeline ID of a workflow topology and deadline to see applying task division expands QoS-guaranteed range without sacrificing cost constraint.

4.2. Experimental Result

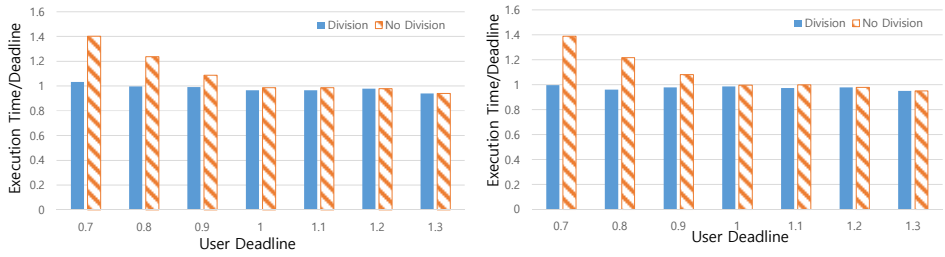
We measured and compared completion time and cost spent to execute different workflows with and without task division.

From **Fig.3** and **Fig.4**, when relatively low deadline is given, we can see that applying task division can execute the request within given deadline while reducing the cost constraint.

5. Conclusion

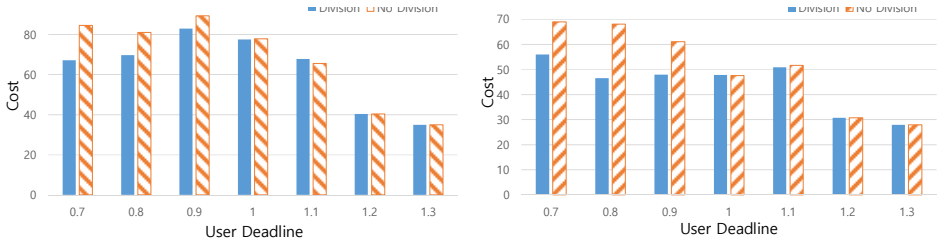
In this paper, we proposed a QoS constrained workflow scheduling scheme using task division which is a kind of the phased scheduling scheme. It has merits in dealing with the uncertainty of task execution time which is changed by the state of the VM resource and finding the near-optimal schedule for 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. We could see that the proposed algorithm provides broader QoS-guaranteed region. Therefore, we



(a) balanced structure (b) unbalanced structure

Fig. 5. Execution time for balanced and unbalanced structure



(a) balanced structure (b) unbalanced structure

Fig. 6. Cost for balanced and unbalanced structure

concluded that the proposed scheme provides broad QoS-guaranteed service region. 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.

References

[1] Yu, Jia, et al., "Workflow scheduling algorithms for grid computing." Metaheuristics for scheduling in distributed computing environments. Springer Berlin Heidelberg, 2008. 173-214.

[2] Menasce, Daniel A., and Emiliano Casalicchio. "A Framework for Resource Allocation in Grid Computing." MASCOTS. 2004.

[3] Xiao, Zhijiao, and Zhong Ming. "A method of workflow scheduling based on colored Petri nets." Data & Knowledge Engineering 70.2 (2011): 230-247.

[4] Yu, Jia et al., "Cost-based scheduling of scientific workflow applications on utility grids." e-Science and Grid Computing, 2005. First International Conference on. IEEE, 2005.

[5] Bharadwaj, Veeravalli et al., "Divisible load theory: A new paradigm for load scheduling in distributed systems." Cluster Computing 6.1 (2003): 7-17.

[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] Kelley Jr, James E. "Critical-path planning and scheduling: Mathematical basis." Operations Research 9.3 (1961): 296-320.

[8] BWA. Available: http://bio-bwa.sourceforge.net/

[9]PharosDreams. Available: http://www.pharosdreams.com/desktop/desktopsolution/home.html