Cost-Aware Workflow Scheduling Scheme Based on Colored Petri-net Model in Cloud

Kyung-no Joo*, Seong-hwan Kim*, Daesun Kim*, Chan-Hyun Youn*

Department of Electrical Engineering, KAIST
Daejeon, Korea
{eu8198, s.h_kim, sundae21, chyoun}@kaist.ac.kr

Abstract. In the science area, workflow management systems are used for the purpose of managing collaborative researches of many organizations. While executing a workflow application, a workflow management system should decide the resource to assign each task and the order to execute the assigned tasks. The processing cost and the completion time can be very different depending on that schedule. Furthermore, users can ask for low processing cost or short completion time. However, satisfying those two requests at the same time is very difficult. Therefore, the existing workflow scheduling schemes try to find the optimal solution while setting bounds to one condition. These schemes can find some schedules that satisfy their own purposes. However, they cannot get the schedules satisfying various SLAs that vary depending on users, workflow applications, and so on. In this paper, we propose the adaptive workflow scheduling scheme based on the colored Petri-Net model which offers two selective workflow scheduling policies. The proposed scheme separates the scheduling phase and the execution phase, and distributes the actual remaining time in the execution phase as well as in the ratio of the processing time processing cost which is being decided for the scheduling phase.

Keywords: Cloud computing, Workflow scheduling, Colored Petri-Net

1 Introduction

Workflow is a means of representing the whole processes of a job and the information paths among its sub-tasks [1]. It is used in the business field at first to define and analyze one's working procedures. Nowadays, not only in the business field, but also in many computer programs which have long or complicated procedures and in computing services which consist of many programs used in each step is workflow used. Scientific applications such as Next Generation Sequencing (NGS) application [2] and Drug Screening application are good example. In order to execute these workflow applications, users need to get assigned to some computing resources and dispatch each program to them. All of these kinds of management functions are performed by workflow management systems. In addition to this, as the cloud computing services being introduced, many researches on enhancing the economic feasibility or the efficiency of workflow management systems are done. Based on the virtualization technology, a cloud computing service provider collects the computing resources in its resource pool, divides the collected resources into the virtual resources whose sizes (number of CPU cores, volume of storage, size of memory) are set as each user wants to have. In addition, the users will be charged for the cloud computing service as much as they used. Therefore, cloud computing can give the service providers the advantage of efficient utilization of their resources and the users the advantage of lowering cost.

While executing a workflow application, a workflow management system should decide the resource to assign each task and the order to execute the assigned tasks. Therefore, workflow scheduling problem plays a key role in WfMS. The processing cost and the completion time can be very different depending on that schedule. Furthermore, users can ask for low processing cost or short completion time, but satisfying those two requirements at the same time is practically impossible. Therefore, the existing workflow scheduling schemes try to find the optimal solution while setting bounds to one condition but they cannot get the schedules satisfying all the various SLAs that vary depending on users, workflow applications, and so on. Workflow scheduling schemes can be divided into three types by their characteristics of scheduling behaviors [3]. *Static scheduling schemes* schedule all the tasks of a workflow instance for once as soon as the instance is initialized. However, it cannot cope with sudden changes of the tasks or resources so it is not practical. *Dynamic scheduling schemes* schedule each task of a workflow instance right before the execution. However, the solutions may not be optimal since it do not considers the whole tasks. Also, there may be some overheads while deciding the resource right before execution. *Phased scheduling schemes* divide the workflow scheduling job and executing job into some phases so that the scheduler schedules and executes the tasks in the same phase in the same way of static scheduling schemes, and after finishing the tasks in a single phase, the scheduler schedules and executes the tasks in the next phase. Therefore, phased scheduling schemes can be considered as the balanced way between the *dynamic scheduling* and the *static scheduling*. Therefore, we adopted the phased scheduling model.

In this paper, we propose the phased workflow scheduling scheme based on the colored Petri-Net model which guarantees the given deadline in a cheap way. Users may choose the policy to utilize. In section 2, we introduce our workflow management system model. We will introduce our scheme in section 3. Then, we will show our experimental result in section 4.

2 Workflow Model

Workflow management system (WfMS) schedules and executes user requested workflow within some constraints such as budget or deadline. Fig 1 shows the architecture and functionalities supported by various components of the workflow management model [4]. At the highest level, users interact with workflow management system by tools such as a workflow designer. They send chemical requests and receives the chemical process result via chemical service layer. User requirements such as deadline or budget are also sent in a service level agreement (SLA) form. In a workflow management layer, it schedules each tasks to proper resources. Resource management layer creates, manages, and terminates the cloud resource. These two layers play an important role in efficient execution of user requests.

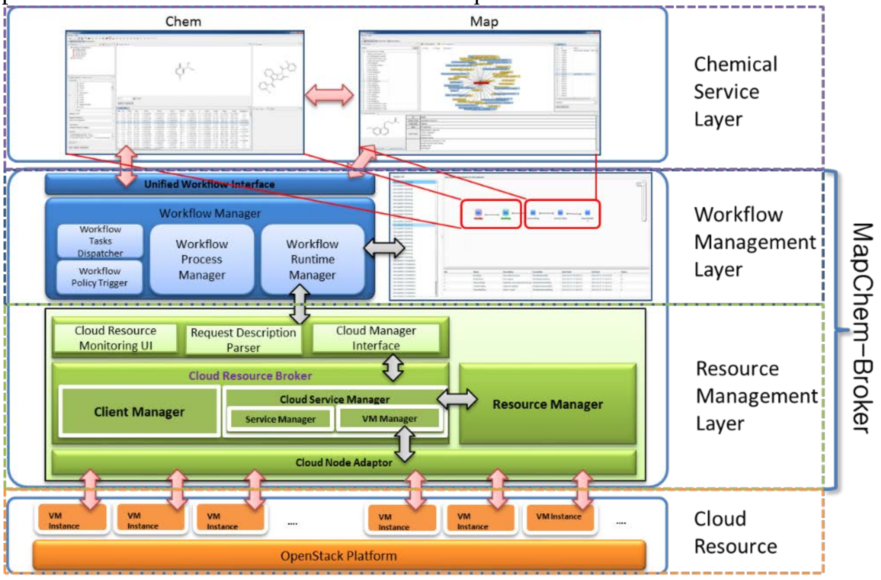

Figure 1 Reference architecture of cloud workflow management system [4]

For each VM type, CSP offers VM specification and its cost information to the WfMS. VM specification means the each VM type's cpu, ram, and storage information. We denote the cpu, ram, and storage of the VM type v be cpu(v), ram(v), and storage(v). Also, we denote the expected execution time of the task T on VM type v be $\mathrm{ET}_{\mathrm{v}}(T)$. We assumed that we pay for leasing VMs in a pay-as-you-go model.

The user request consists of workflow topology W and the service level agreement SLA. We used colored Petri-net model in order to represent the workflow topology W. A Petri-net is one of the mathematical tools for representing workflow topologies. A Petri-Net is defined as a 4-tuple $PN = (P, T, A, M_0)$, where P is a finite set of places, T is a finite set of transitions, A is a finite set of arcs, and M_0 is an initial marking of tokens [5]. We call the Petri-net model colored Petri-net when the tokens are classified by "colors". In our work, we separated the token by two colors: *scheduling token* and *execution token*. *Scheduling token* first scans the entire workflow topology and determines the value that helps scheduling. *Execution token* maps the task onto the VM and monitors the process. Initially, *scheduling token* is located at the last task of the workflow and *execution token* is located at the first task of the workflow.

We assume that each places contain remaining execution time information *ret*. The remaining execution time of the place p (*ret(p)*) is the expected execution time to finish execution of the rest of the workflow from p. Since we do not know the execution time, we used average execution time for all VM types to calculate *ret*. Also, we assume that each transitions contain expected proportion of the task *prop*. The expected proportion of the task t (*prop*(t)) is the importance of the current task's expected execution time compared to the expected execution time of the entire workflow. These two values are determined by *scheduling token* and will be used in scheduling process. Also, we assume each tokens have information of remaining time in order to map each tasks onto proper resources.

3 Proposed Phased Adaptive Workflow Scheduling Scheme

Our scheme adopted phased scheduling model, which improves the weak point of static scheduling and dynamic scheduling. There are two phases in our scheme: *scheduling phase* and *execution phase*. The purpose of the scheduling phase in our scheduling scheme is to find the expected proportion of the time consumption of each task to the time consumption of all the tasks waiting to be processed. In the execution phase, the proposed scheduling scheme decides the amount of computing resources to be assigned to a task as the proportion calculated before in the scheduling phase, so that it can make a workflow task schedule that reflects the actual margin of the given constraint.

3.1 Scheduling Phase

When a user sends workflow request to WfMS, WfMS first puts two tokens: one scheduling token at the end place, and one execution token at the entry place. In scheduling phase, the scheduling token recursively scans backward to the entry task and determines *ret(p)* of each places and *prop(t)* of each transitions.

The scheduling token moves backward to a new place and we can determine *ret* and *prop* value while moving step by step until it arrives the start point of the workflow. Traditional Petri-net model do not describes the behavior of token moving backward. However, in our work, we assumed that token may move backward in the opposite way to the token firing to forward. Fig 2 shows the movement of tokens in traditional Petri-net model. In traditional Petri-net model, when token fires, current token at the place disappears and a token is created at the places that come after. This process is shown in Fig 2(a) and Fig 2(b). Token fires when a firing condition is satisfied: when there appears an AND-join pattern, tokens should wait until each places which are located before to the place that all branch joins have one token as in Fig 2(c) and Fig 2(d). In our model, scheduling token moves backward in a same way. We can easily see Fig 2 in an opposite way. When an AND-join pattern appears as in Fig 2(d), current token disappears and all places that tokens are generated at the places that come before in Fig 2(c). Also, when an AND-split pattern is appeared, we should wait until first place of each branch has token.

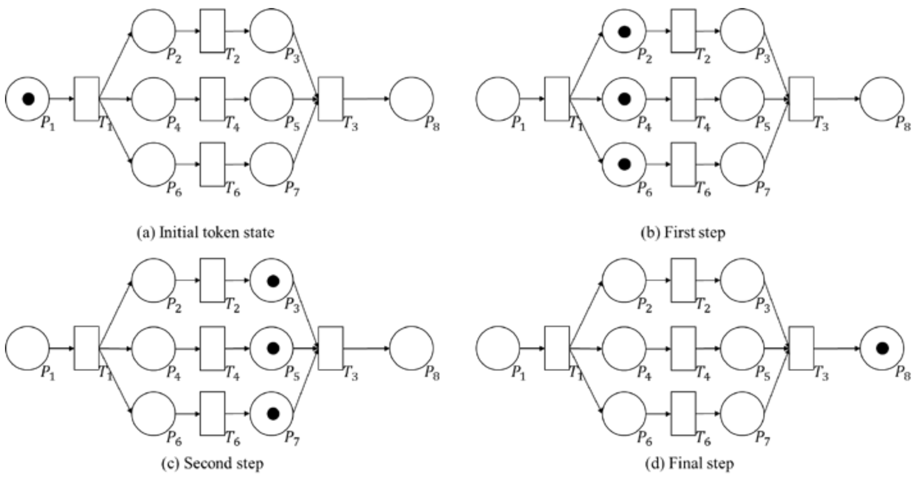

Figure 2 Example of token firing process in traditional Petri-net model

Initially, the scheduling token locates at the last place of the workflow and the remaining execution time of the last place is definitely zero since there is no transitions (tasks) left. Let place where scheduling token exists be *p* and the next transition and the next place be p^* and p^{**}. Then, the remaining execution time of the place p can be determined in a recursive way by equation **(1)**.

$$ret(p) = \begin{cases} ret(p^{**}) + \frac{1}{k} \sum_{v}^{VM\ types} ET_v(p^*) & if\ p^{**}\ exists \\ 0 & if\ p^{**}\ do\ not\ exist \end{cases} \quad \textbf{(1)}$$

We do know exactly know the execution time since VM types are not scheduled yet. Therefore, we used the average execution time of p^* for all VM types. Also, the expected proportion of the task that comes after the place p can be determined by eq.**(2).** *prop*(t) is the ratio of the execution of the task to the remaining execution time so it can be simply obtained by the expected execution time of the task divided by the *ret* of p.

$$prop(\mathrm{p}^*) = \frac{\frac{1}{k}\sum_{v}^{VM\ types} ET_v(p^*)}{ret(p)} \quad \textbf{(2)}$$

When a scheduling token arrives at the entry task, it means that *ret* and *prop* of all places and transitions are determined. Then, the scheduling phase ends and moves to the execution phase.

3.2 Execution Phase

In execution phase, execution token moves forward from entry node to end node, dynamically scheduling and executing tasks. Token firing condition is the same as the traditional Petri-net model but it moves after the next transition (task) is completed. Our scheme tries to schedule tasks in the cheapest way while guaranteeing the given deadline. The basic idea of our scheduling scheme is to allocate the deadline of a task considering the importance of the task. When a certain task needs high computing resources compared to other task, we map the task to a high computing resource. Proportion of a task *prop(t)* shows the importance of the task compared to the rest. Therefore, we consider *prop(t)* and allocate the portion of resources in order to ensure deadline.

Each execution tokens accompany the remaining deadline information. We denote the remaining deadline of execution token at place *p* be *p.rd*. For example, when execution token is located at the entry place p_e, we can easily know that $p_e.rd$ is the total deadline of the workflow request. To schedule the next task, we defined sub-deadline concept. sub-deadline is the deadline of the task t and it can be obtained by simply multiplying the remaining time and the prop value of p as depicted in equation **(3)**.

$$deadline(p^*) = p.rd \times prop(p^*) \tag{3}$$

We assumed that the WfMS have information of expected execution time of tasks on every VM types. The aim of deadline policy is to guarantee the deadline while trying to use resource in a cheap way. Therefore, we can finish scheduling by finding the cheapest VM type which assures the sub-deadline. Token stays until the execution of p^* is completed and the next place is ready to be started. When triggered, it fires to the next places while the remaining time value is decreased by actual execution time of p^*. Algorithm 1 summarizes the entire process of scheduling phase and execution phase.

Algorithm 1. Phased scheduling scheme
Input: W: workflow topology in a Petri-net form, SLA: deadline constraint
Output: execution result

```
※ Scheduling phase
while true do
    Let SP = $\{P_1, P_2, \dots, P_n\}$ be places which has scheduling tokens in current
    if SP has only one element at entry node then
        end scheduling phase
    end if
    for p = each element in SP do
        determine $prop(p^*)$ and $ret(p)$ by eq.(1) and eq.(2)
        if all members of $({}^{**}p)^{**}$ have scheduling token then
            delete p in SP
            add $({}^{**}p)$ in SP
        end if
    end for
end while
※ Execution phase
while true do
    Let EP = $\{P_1, P_2, \dots, P_n\}$ be places which has execution tokens in current
    if EP has only one element at entry node then
        end execution phase
    end if
    for $p$ = each element in EP do
        if $p^*$ is schedulable (each $^*(p^*)$ have one execution token) then
            $\text{deadline} = \text{p.rt} \times \text{prop}(\text{p}^*)$
            Let V be the cheapest VM type to execute $\text{p}^*$ within the deadline
            Execute $\text{p}^*$ at V
            $\text{p}^*.rt = p.rt - actual\ execution\ time\ of\ p^*$
        end if
    end for
end while
```

4 Experiments and Evaluation

4.1 Experiment Environment

The experimental environment consists of workflow designer, workflow broker, and OpenStack Cloud as shown in Fig. 3. We defined three workflow applications using workflow designer, and requested execution of one workflow within a certain deadline. The task request including the SLA information was sent to the workflow broker, and then a workflow broker did utilize the deadline information and allocated each sub-task within the sub-deadline. The workflow management system used in the experiment was implemented for OpenStack cloud so that it managed VMs under OpenStack environment to run each sub-task.

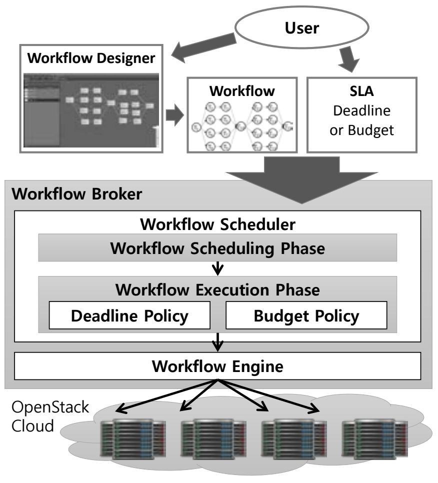

Figure 3 Experimental environment

Three examples of workflow applications are shown in Fig. 4. Workflow type 1 (Fig. 4(a), and (b)) has some sequentially connected split-merge pairs that have some parallel tasks. Workflow type 2 (Fig. 4(c), and (d)) has only two split-merge pairs. However, they have lots of parallel tasks. Workflow type 3 (Fig. 4(e), and (f)) are composed of hybrid structures of workflow type 1 and the type 2, because its outer split-merge pair has many parallel tasks, as well as its inner split-merge pairs are connected sequentially and show characteristics in some parallel tasks. We used task type 1 and type 2 in distribution. The shaded area of Fig 4(b), (d), and (f) are task type 2 and rest of others used task type 1. Workflow type 1 is appeared in Mao's work[6] and workflow type 2 and workflow type 3 are show in Jia Yu's work[7] in experiments. Topologies used here are exactly same as the ones used in previous works' experiment. However, tasks are different from them.

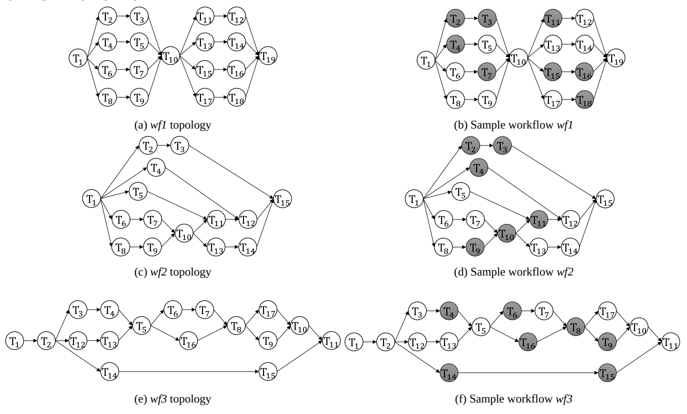

Figure 4 Three types of workflow topology used in experiment

4.2 Experimental Results

We requested the execution of each workflow applications repeatedly with various SLAs (deadlines), and investigated the actual execution time and cost. Additionally, we 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 the given deadline. Whether the proposed scheme yield the optimal schedule can be verified by comparison with other scheduling policies and other scheduling schemes, but it is not covered in this paper.

Fig 5 shows the experimental results according to the deadline policy used in the proposed scheme. Fig 5(a). (c), and (e) represents the actual execution time with different workflow types versus increment of deadline. Bold lines in Fig 5(a), (c), and (e) represents the actual processing time of each workflow. The gray lines stands for the differences in QoS information of the applications. We can find that three graphs show the similar result. When the deadline is too low, the deadline policy cannot meet the deadline although the broker allocates whole computing resource with large VM types. Therefore, there are some QoS differences in low deadline requirement. However, when the deadline is given adequately, the broker can schedule for adequate VM types namely, the deadline policy assures the requirement. Results show that the deadline is always guaranteed. In advance, Fig 5(b), (d), and (f) shows the execution costs of different workflow types when we change the deadline. All graphs are monotone decreasing. When the deadline is small, larger type flavor are used more frequently in order to meet deadline and the execution cost tends to be more expensive. In the same manner, the execution cost tends to be cheaper when the deadline is large. Therefore, we can conclude that the deadline policy schedules workflow with guaranteeing the deadline, while trying to use minimum resources. We checked this policy works well in various types of workflow.

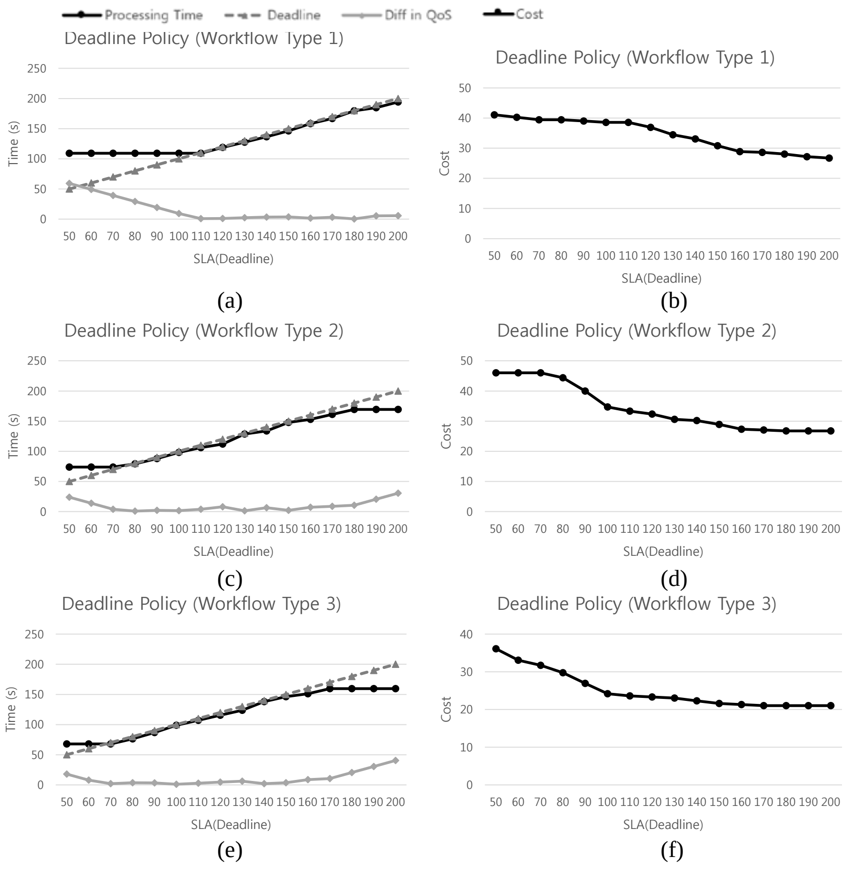

Figure 5 Experimental result

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 user given. Our model uses the phased scheduling model so that it can schedule dynamically with low complexity and close to optimal. 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.

Acknowledgement

This work was supported by the ICT R&D program of MSIP/IITP. [10038768, The Development of Supercomputing System for the Genome Analysis]

References

1. Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers, "Examining the challenges of scientific workflows," Computer, vol. 40, pp. 24-32, December 2007.
2. REIS-FILHO, Jorge S. “Next-generation sequencing. Breast Cancer Res”, 2009, 11.Suppl 3: S12.
3. Zhijiao Xiao, and Zhong Ming (2011), “A method of workflow scheduling based on colored Petri nets,” Data & Knowledge Engineering, vol. 70, issue 2, pp. 230-247.
4. Ye Ren (2012), “A Cloud Collaboration System with Active Application Control Scheme and Its Experimental Performance Analysis”
5. VAN DER AALST, W. M. P. (1998). “THE APPLICATION OF PETRI NETS TO WORKFLOW MAN-AGEMENT.” Journal of Circuits, Systems and Computers 08: 21–66.
6. Mao, Ming, and Marty Humphrey. (2011). “Auto-Scaling to Minimize Cost and Meet Application Dead-lines in Cloud Workflows.” 2011 International Conference for High Performance Computing, Network-ing, Storage and Analysis (SC): 1–12.
7. J. Yu and R. Buyya, "A taxonomy of scientific workflow systems for grid computing," SIGMOD Rec., vol. 34, pp. 44-49, September 2005.