ISSN(Print): 2287-5905

http://dx.doi.org/10.3745/KTSDE.2012.1.3.137

A Cost-Efficient Job Scheduling Algorithm in Cloud Resource Broker with Scalable VM Allocation Scheme

Ye Ren[†] · Seong-Hwan Kim[††] · Dong-Ki Kang[†††] · Byung-Sang Kim[††††] · Chan-Hyun Youn[†††††]

ABSTRACT

Cloud service users request dedicated virtual computing resource from the cloud service provider to process jobs in independent environment from other users. To optimize this process with automated method, in this paper we proposed a framework for workflow scheduling in the cloud environment, in which the core component is the middleware called broker mediating the interaction between users and cloud service providers. To process jobs in on-demand and virtualized resources from cloud service providers, many papers propose scheduling algorithms that allocate jobs to virtual machines which are dedicated to one machine one job. With this method, the isolation of being processed jobs is guaranteed, but we can't use each resource to its fullest computing capacity with high efficiency in resource utilization. This paper therefore proposed a cost-efficient job scheduling algorithm which maximizes the utilization of managed resources with increasing the degree of multiprogramming to reduce the number of needed virtual machines; consequently we can save the cost for processing requests. We also consider the performance degradation in proposed scheme with thrashing and context switching. By evaluating the experimental results, we have shown that the proposed scheme has better cost-performance feature compared to an existing scheme.

Keywords : Job Scheduling, VM Allocation, Cloud Computing, Request Placement, Workflow Management

클라우드 자원 브로커에서 확장성 있는 가상 머신 할당 기법을 이용한 비용 적응형 작업 스케쥴링 알고리즘

Ye Ren[†] · 김 성 환[††] · 강 동 기[†††] · 김 병 상[††††] · 윤 찬 현[†††††]

요 약

사용자들은 자신의 작업을 처리하기 위해 자신에게만 한정된 가상 컴퓨팅 자원을 클라우드 서비스 제공자로부터 할당 받아 타 사용자로부터 독립된 환경에서 작업을 처리하게 된다. 이를 자동화된 방법으로 최적화를 대신 수행해주기 위한 모델로 브로커 미들웨어가 제시되었고 마감시간을 만족하는 이내에서 자원 이용률을 높이는 접근법으로 필요 가상 머신의 숫자를 줄여 비용을 절약한다. 이를 다루는 많은 논문들에서 작업 스케줄링은 기존 사용자들간의 독립을 보장하여 하나의 가상 머신이 하나의 작업에 한정된 가상 머신에서 처리하는 방식으로 다루어지고 있다. 하지만 기존의 SRSV 방식에서는 높은 정도의 다중 프로그래밍 작업이 아닐 경우 시스템을 효율적으로 사용하지 못한다. 이에 본 논문에서는 해당 자원을 마감시간과 스래싱(thrashing), 문맥 전환(context switching)에 따른 성능 저하를 고려한 상태에서 다중 프로그래밍 정도를 높여 낭비되는 자원을 최소화하여 비용을 절약하려고 한다. 실험 결과를 통해 제안하는 방법이 제약조건 이내에서 기존의 방식에 비해 좀 더 좋은 가격 대비 성능을 가지는 것을 보인다.

키워드 : 작업 스케줄링, 가상머신 할당, 클라우드 컴퓨팅, 요청 할당, 워크플로우 관리

※ This research was equally supported by R&D programs of MEST/NRF [2012-0020522, the Next-Generation Information Computing Development Program], and MKE/KEIT [10039260, Integrated development environment for personal, biz-customized open mobile cloud service and Collaboration tech for heterogeneous devices on server].
※ This research also was supported by IT R&D program of MKE/KEIT [10038768, The Development of Supercomputing System for the Genome Analysis].

† 준 회 원 : 한국과학기술원 그리드미들웨어연구센터 위촉연구원
†† 준 회 원 : 한국과학기술원 전기및전자공학과 석사과정
††† 준 회 원 : 한국과학기술원 전기및전자공학과 박사과정
†††† 준 회 원 : 한국과학기술원 정보통신공학과 박사과정
††††† 종신회원 : 한국과학기술원 전기및전자공학과 교수
논문접수 : 2012년 11월 22일
심사완료 : 2012년 12월 3일
* Corresponding Author : Chan-Hyun Youn(chyoun@kaist.ac.kr)

1. Introduction

To utilize computing resources efficiently in enterprise environment, cloud computing provides idle resources to outside of company with separating a physical machine into multiple logical and isolated virtual machines based on virtualization technology. To endure burst workload from users, companies have to prepare resources more than expected value of demand. The increasing idle time of resources lead to the resource utilization rate lower than 20%. To utilize idle resources efficiently, server provisioning was proposed [1] and it becomes cloud service with virtualization technique. One of the cloud computing service forms is Infrastructure as a Service (IaaS) and it provides infrastructure resources by configuring the server, storage, and network into virtualized resources reacting to on-demand requests. In the view of cloud service users, the cloud service reduces the initial purchasing price of computing resources and server management price. Also users can easily handle the unexpected and dynamic requests of resources [2]. Users have to provide information about resource type which contains number of CPU cores, memory capacity and storage capacity and the resource price is dependent on the type of the resource. Generally, a more powerful resource costs more and the cloud provider charges for each provided virtual machine in full-hour billing model, so it is necessary to release the virtual machine if it is not serving the job [3]. Because users have to consider the type of resource based on the deadline and completion time of jobs and have to passively request virtual machine and terminate virtual machine based on full-hour billing model, users need to have systematic knowledge and there can occur mistakes or waste. To solve these problems, the broker is proposed which mediates between cloud service providers and service users with users' requirements and related policies. So users can manage the jobs and resources in an efficient way. Broker has requirements and one of them is job scheduling which deals with the problem that how can virtual machines serve the jobs in order to complete all jobs within deadline and minimize overall cost by reducing the number of needed VMs through enhancing the resource utilization.

Previous papers [3, 4] focused on the isolation between multiple users. So a VM is dedicated to one job and we will call it as Single Request per Single VM (SRSV). But SRSV manner can't use resources in maximum utilization because of low degree of multiprogramming. Also there is another way of job scheduling which allocates a VM to process multiple requests. But they calculate the limitation based on simply adding up the job workload and not consider the profit of increased utilization based on parallelization. [5].

In this paper, we increase the degree of multiprogramming in proper range with multi-thread or multi-tasking method to minimize the wasted resource and maximize the utilization of resource. The Multi Request per Single VM (MRSV) policy which shares a VM among multiple jobs can maximize the system utilization and satisfy the deadline if the deadline of job is not sensitive.

2. Problem Description

2.1 Conventional Workflow Management Systems

Several workflow management systems have been developed during the last two decades. Some of them are just doing workflow management while others have the resource management functions. Kepler [6] is a software application for the analysis and modeling of scientific data. It simplifies the effort needed to create executable models by using a visual representation of these processes (user-friendly graphical user-interface). These representations for scientific workflows display the flow of data among discrete analysis and modeling components. Kepler allows scientists to create their own executable scientific workflows by simply dragging and dropping required components onto a workflow design panel. Then user can connect the components to construct a specific data flow, and finally create a visual model of the analytical portion of their research. Kepler represents the overall workflow visually so that it is easy to understand how data flow from one component to another. It models a workflow system as a composition of independent components (actors) that communicate through well-defined interfaces. Kepler includes distributed computing technologies which allow users to share their data and workflows with other scientists and to use data and analytical workflows from others around the world.

Triana [7] is an open source problem solving environment developed at Cardiff University that combines an intuitive visual interface with powerful data analysis tools. Triana provides a visual programming interface with functionality represented by units. Applications are written by dragging the required units onto the workplace and connecting them to construct a

workflow. Triana PSE (Problem Solving Environment) allows the graphical composition and distributed execution of services. Pegasus [8] (Planning for Execution in Grids) is a workflow management engine developed and used as part of several projects such as GriPhyN [9]. Pegasus enables scientists to construct workflows in abstract terms without worrying about the details of the underlying Cyber Infrastructure.

2.2 Conventional Workflow Scheduling Algorithm Heuristics

In general, workflow scheduling problem is usually considered as NP-complete problems. Thus, even though the DAG scheduling problem can be solved by using exhaustive search methods, the complexity of generating the schedule becomes very high. As workflow scheduling is an NP-hard problem, heuristic and meta-heuristic based scheduling strategies are applied to achieve near optimal solutions within polynomial time. In the following, we discuss some of the well-known heuristics and meta-heuristics for workflow scheduling in Cloud. Each heuristic is attempting to minimize the makespan. According to the description of papers [10, 11], we summarize heuristics as follows.

Greedy: The Greedy is literally a combination of the Min-min and Max-min heuristics. The Greedy heuristic performs both of the Min-min and Max-min heuristics, and uses the better solution.

HEFT: Heterogeneous Earliest Finish Time (HEFT) [4] is a well-established list scheduling algorithm, which gives higher priority to the workflow task having higher rank value. This rank value is calculated by utilizing average execution time for each task and average communication time between resources of two successive tasks, where the tasks in Critical Path get comparatively higher rank values. Then it sorts the tasks by decreasing order of their rank values and the task with higher rank value is given higher priority. In the resource selection phase, tasks are scheduled in the order of their priorities and each task is assigned to the resource that can complete the task at the earliest time. The advantage of using this technique over Min-Min or Max-Min is that while assigning priorities to the tasks it considers the entire workflow rather than focusing on only unmapped independent tasks at each step.

2.3 Definition of the Framework for Workflow Scheduling in Cloud

In the proposed framework, for each workflow application execution, there are three tightly interactive roles – they are users, the broker, and the service providers. A workflow application is submitted to the cloud by users, scheduled by the workflow scheduling scheme based on the interaction between the broker and the resource providers, and finally executed by the resources leased from the service providers. So, there is no direct interaction between the users and the service providers, which are decoupled by the broker. Thus, it is required to provide schemes or tools that allow the three roles to participate in the computing work. The three roles need to express their requirements and facilitate scheduling decisions to fulfill their objectives. We therefore utilize SLA (Service Level Agreement) that is usually defined in the community as a business agreement between two of them to create the common understanding about services, responsibilities, and others. There will be two bipartite SLAs that are represented by the SLA type I relation (SLA_1) which is established between users and the broker, as well as the SLA type II relation (SLA_2) which is established between the broker and providers. In later part of this paper, the SLA_1 may be used interchangeably with the SLA. The two bipartite SLA's values together define the quality of service offered to a user. The broker incorporates a QoS-enabled workflow management module and an adaptive resource provision module. The workflow management module will perform a mapping function from the SLA_1 to a corresponding SLA_2. QoS constraints declared in SLA_1 from user's perspective will be mapped to resource parameters declared in SLA_2 which is from a resource provision module's perspective. Resource provision module will then use the SLA_2 to allocate a suitable resource to execute the current sub-task in the workflow.

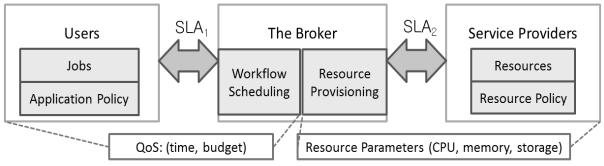


Fig. 1. Important roles of job scheduling in the cloud

Definition 1 (workflow): Within a DAG workflow model, let T be the finite set of tasks $T_i(1 \le i \le N)$ in the workflow, and Φ be the set of directed edges of the form (T_i, T_j), where T_i is called the immediate parent task of T_j, and T_j is the immediate child task of T_i, respectively. A child task cannot be executed until all of its parent tasks have been completed execution. So, a workflow is denoted as $W(T,\Phi)$.

Definition 2 (constraint workflow): From the users' perspective, they hope the workflow they submitted can be finished execution within some specified deadline or budget. For example, if users will specify deadline constraint D, that is to say, the user want to run his or her workflow application no later than a specified deadline. Meanwhile, the user hopes that the workflow should be finished with the least possible cost. In such case, the workflow application can be described as a tuple $W(T,\Phi,D)$.

Definition 3 (user's application/task): The characteristic of a task T_i consists of: {type, length (L_i in Million Instructions), input file(s) ($I_i^1, I_i^2, ..., I_i^{k_i}$, where k_i is the total number of all input files for task T_i), output file(s) ($O_i^1, O_i^2, ..., O_i^{l_i}$, where l_i is the total number of all output files for task T_i)}. The tuple that represents a task is $T_i = L_i, [I_i]_{k_i}, [O_i]_{l_i}$.

Definition 4 (cloud VM type): A VM type r_j can be modeled with six parameters: {number of compute units (can be transferred into Million Instructions per Second) r_{c_j}, network speed (bit rate, bit/sec) r_{n_j}, memory size (GBs) r_{m_j}, storage space (GBs) r_{s_j}, charging policy ($/hour) r_{p_j}, the latency time after it being launched and before being ready to use (sec) r_{l_j}}. The tuple that represents a VM instance: $r_j = [r_{c_j}, r_{n_j}, r_{m_j}, r_{s_j}, r_{p_j}, r_{l_j}]$

Definition 5 (The goal of the broker): to apply workflow scheduling mechanism and resource auto-scaling scheme to finish all submitted requests within users' specified deadlines and also try to minimize the total cost for purchasing the processing resources in the cloud.

Table 1. Parameters in job-resource mapping

Parameter Name	Meaning
N	The total number of tasks within a workflow
$D(T_i)$	The assigned executing time of task T_i within the workflow
R_i	The set of all the VM types which are capable of executing a particular task T_i (that is, can finish the task within its deadline) in the broker's current VM pool.
m_i	The total number of VM types in set R_i currently available in the broker's VM pool.
r_i^j	Represent that the VM type of r_j is assigned to execute the task T_i. $(r_i^j \in R_i, 1 \le i \le N, 1 \le j \le m_i)$

Definition 6 (Parameter definitions): we make additional definitions in Table 1 which are involved in the mapping relationship between a task and an executing VM type.

2.4 Generic Procedures for Scheduling a Workflow in Cloud

An exact estimation of the performance for each task on each VM type is critical for the later scheduling decisions. The broker can estimate the performance of a task on different types of VMs, by using below performance estimation technique.

Lemma 1 (performance estimation of each task): For running task T_i on VM type r_j, the estimated execution time $T_{r_j}^{ee}(T_i)$ (in hours) is given by Eq. (1).

$$T_{r_j}^{ee}(T_i) = T_{\text{transfer}} + T_{\text{execution}} = [(\sum_{1 \le n \le k_i} I_i^n \cdot d / r_{n_j}) + (L_i / r_{c_j})] / T \quad (1)$$

Proof: The estimated execution time of a task on a VM consists of two parts of time, the data transfer time and the application running time. The application cannot start to be processed on the computing resource before all necessary input data is transferred to the computing resource. The data transfer time can be calculated from the total size of input files to be transferred and the bandwidth of the VM type; and the application running time can be decided by the length of the application and the computing speed of the VM type. We time the total input size by d which is the data rate and equals to 8 in later experiments, since in our previous definition, the size of input files is defined in the unit of Bytes while the bandwidth of a VM type is defined in the unit of Bit per second. Since the cloud resources will be charged by instance-hour, we transfer the value of time from the unit of second to the unit of hour which yields to that the T in Eq. (1) equals 3600.

By using Lemma 1, with different VM types we may get a set of values for $T_{r_j}^{ee}(T_i)$.

We will also get the averaged estimated execution time for running task T_i by Eq. (2).

$$T^{ee}(T_i) = \frac{1}{m_i} \sum_{1 \le j \le m_i} T_{r_j}^{ee}(T_i) \quad (2)$$

The minimum estimated execution time for running task T_i is calculated by Eq. (3).

$$T_{\min}^{ee}(T_i) = \min\{T_{r_j}^{ee}(T_i) \mid 1 \le j \le m_i\} \quad (3)$$

The corresponding estimated execution cost $C^{ee}_{r_j}(T_i)$ (in $) for running task T_i on VM type r_j is given by Eq. (4).

$$C^{ee}_{r_j}(T_i) = \left\lceil T^{ee}_{r_j}(T_i) \right\rceil \cdot r_{p_j} \tag{4}$$

With the deadline of each task, we will allocate a VM type for executing each task by using Theorem 1.

Theorem 1 (VM type allocation): Select the VM type r_j for running task T_i , subject to the condition that

$$\{T^{ae}_{r_j}(T_i) \le D(T_i)\} \&\& \{C^{ae}_{r_j}(T_i) = \min\{C^{ae}_{r_j}(T_i) \mid 1 \le j \le m_i\}\}.$$

Proof: Assume that we know the assigned execution time for each task $D(T_i)\,(1 \le i \le N)$ within the workflow. During run-time in actual situation (all actual input files and their sizes are known), we can compute the cost of every available resource type in set R_i for task T_i as shown in Eq. (5) and Eq. (6).

$$T^{ae}_{r_j}(T_i) = T_{\text{transfer}} + T_{\text{execution}} = \Big(\sum_{1 \le n \le k_i} I^n_i \cdot \Big/ r_{n_j}\Big) + (L_i / r_{c_j}) \tag{5}$$

$$C^{ae}_{r_j}(T_i) = \left\lceil T^{ae}_{r_j}(T_i) \right\rceil \cdot r_{p_j} \tag{6}$$

The scheduler in-time sorts all the costs from the available services and allocates the service that can both meet the $D(T_i)$ and also has the minimum charge for the considered task. That is, to pick the resource type r_j subject to

$$\{T^{ae}_{r_j}(T_i) \le D(T_i)\} \&\& \{C^{ae}_{r_j}(T_i) = \min\{C^{ae}_{r_j}(T_i) \mid 1 \le j \le m_i\}\}$$

(*q.e.d*).

3. Multi Requests per Single VM

After the resource type decision, in order to allocate user's requests to VM instances with maximization of VM instance utilization, we allocate multi requests to each VM instance. In most of the existing papers, one VM is allocated for processing a single request - Single Request Single VM (SRSV). In this paper, the big difference is that we enable each VM instance to process multiple requests by means of multiprogramming - Multiple Requests Single VM (MRSV) as shown in Fig. 3. By enhancing the CPU utilization as well as maximizing the VM resource utilization, processing the same total amount of requests will need fewer number of VM instances. Since the broker will need to lease fewer VM instances from cloud service providers to process user's requests, the broker can save the cost.

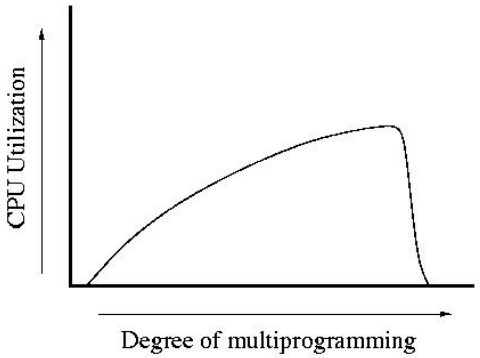


Fig. 2. Relationship between multiprogramming and CPU utilization [12]

Fig. 2 shows that multiprogramming enhances the CPU utilization within some edge range. This is because that among the many operations that interact with CPU, the I/O is the most time-consuming one and I/O speed is much slower than the CPU speed. If one CPU handles only one process or one thread, it will probably spend much time waiting to get data from I/O and only spend little time doing the real processing after loading the data in virtual memory. As a result, the CPU utilization is low. While for the multiprogramming, in particular in this paper the multiple threads technology, for each thread, CPU needs to wait for data I/O as well; however, it is very probable that instead of pure and long-time waiting for data I/O of one thread, the CPU can do the data processing for another thread. In other words, the CPU gets enhanced its utilization by transferring the waiting time for thread A to the processing time for thread B [13]. Though this is the virtue of multiprogramming, we need to be careful not to overplay this trick, because once the degree of multiprogramming exceeds some limit, the CPU utilization will degrade dramatically. This is because too much multiprogramming leads to thrashing and the CPU spends more time handling page faults than it should spend in handling our desired work [14, 15]. In sum, an appropriate degree of multiprogramming will give a strong probability that the CPU will get its utilization enhanced.

Detailed illustration to compare the SRSV and MRSV is shown in Fig. 3. The above part of figure shows the

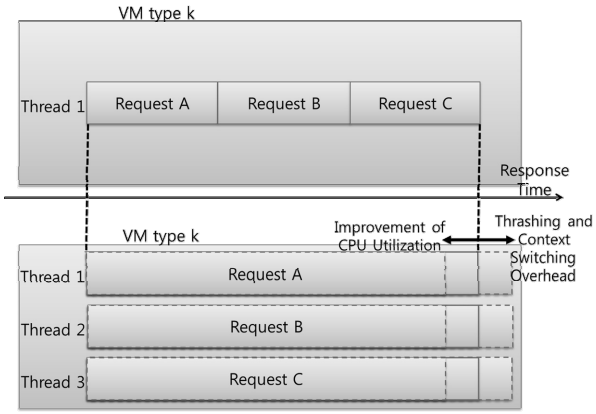


Fig. 3. SRSV vs MRSV

scenario where one VM instance processes requests one by one using one thread; the below part of figure shows that there are multiple threads in one VM instance and the requests are processed in parallel. In MRSV scenario with a proper degree of multiprogramming, we can expect a shortened overall completion time than one VM instance dealing with the same amount of requests in the SRSV scenario. On the contrary, in MRSV scenario with over-degree of multiprogramming, we can expect a deteriorated performance as compared with the SRSV.

An experiment for practically comparing the SRSV and MRSV on one VM instance is conducted and the results are shown in Fig. 4. To get the experimental result, we use simple chemical application which calculates the chemical properties. We allocate two VMs each with 1 Core, 1GB RAM, 10GB Storage and make VMs to process jobs in sequential and in parallel ways. We can see that with the number of jobs to be processed for one VM instance under eight, the MRSV demonstrates a shorter overall completion time than the SRSV.

After the VM type of user's request is selected, the VM allocation module checks the current number of being processed requests in each VM instance of the selected

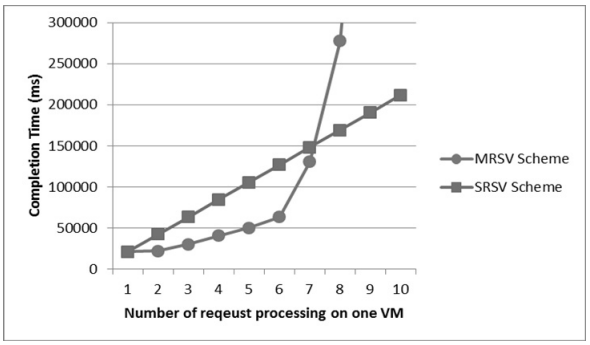


Fig. 4. Completion time comparison of SRSV and MRSV under different number of jobs processed by a VM instance

type. The upper limit threshold of allocation capacity to each VM type is the value obtained from many empirical tests in achieving the maximization of resource utilization profit and getting the point where the subtraction of MRSV and SRSV is maximized. The goal is to choose the instance which is processing the largest amount of workload among all instances and also the amount of the being processed requests is under the instance's upper limit of request amount. This is to utilize the resource capability to the fullest of each VM instance and also satisfy the user's requirement.

The description of the procedures for resource type decision and VM instance selection is shown in below Section 5 in more detail.

4. MapChem-Broker for Chemical Application

An adaptive cloud resource broker named MapChem-Broker integrates the functionality of the workflow management system and the resource management system. As shown in the Fig. 5., The broker mediates in the middle between the application layer and the resource layer and it is designed as layered architecture including Chemical Service Layer (CSL), Workflow Management Layer (WML), Resource Management Layer (RML), and Cloud Resource Layer (CRL).

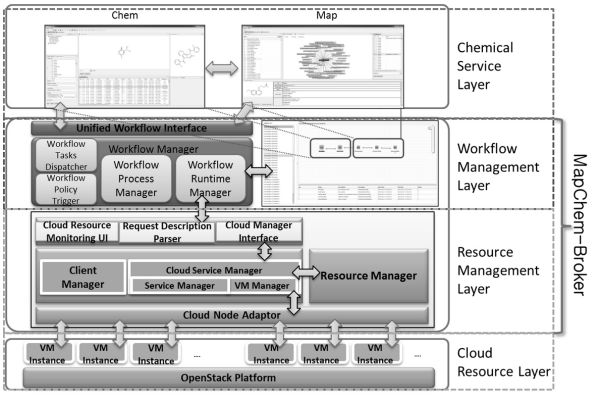


Fig. 5. Layered architecture of MapChem-Broker

On CSL, Chem service and Map service are available. The Chem service provides calculation of molecular descriptors and expected QSAR results by using several equations and more regression models such as multi linear regression, generic algorithm and principle component analysis. Map service provides related articles for drug repositioning and the information of chemical compounds on the market which are similar to the chemical compounds that user want to regenerate. Map

and Chem services are provided to users through convenient web-based graphical user interface. Many users can access the system to operate Map and/or Chem services simultaneously. Users design the chemical integrated applications in the form of a workflow by using MapChem application designer and submit the workflow to the Unified Workflow interface in the Workflow Management Layer as a HTTP request.

On WML, the execution of Map and Chem services can be scheduled to virtualized cloud resources which are involved in our whole system. By using workflow management system, workflows composed of Map and Chem services are divided into sub-tasks, dependencies, and other relative parameters which are represented as XML description files and later dispatched to virtual machine instances of the Cloud infrastructure. Through this layer, user's SLA expectations can be translated into the corresponding resource property specifications which can be used as selecting standard in the resource allocation process. By the combination of the WML and the RML, user's SLA specifications can be guaranteed.

Compared to the case which uses physical machines to process requested application, the adaptive utilization of virtual machine instances has advantages such as efficiency of resource utilization, and the scalability for resource extension according to dynamic fluctuation of total number of users' requests. The RML is in charge of virtual machine management which means the assigning of a VM instance in the Cloud infrastructure to a sub-task within the workflow, to provide resource monitoring, and to manage resource scheduling policies. The CloudNode adapter is an interface between the Resource Management Layer and the virtualized cloud resources. Resource requests for the same VM resource type will go into the queue of that type and wait to be processed. To allocate for the sub-task of workflow/job a proper VM server subjective to optimal performance, the SLA has to be guaranteed by allocating the resource that has the requested resource capability parameters obtained from the SLA translation in the Workflow Management Layer. The information of Cloud resource monitoring is reported to users periodically.

For the CRL, we set up open-source Cloud platform called OpenStack which supports a variety of hypervisors such as XEN, KVM, and so on. We use the component naming Nova in the OpenStack which provides computing operations to users. Nova manages VM assignment, release, and monitoring. In order to process user's sub-jobs, it is necessary to establish the environment to enable the execution of the specific application. Target VM instance has to be configured with proper Operating System, H/W specification, development environment, and several other indispensable utilities. There are images for launching a proper VM instance and Nova is used to configure these images. By using predefined image with specified user's requirements, the work of user can be operated on VM instances launched from that image. Fig. 6. illustrates the working procedures inside the MapChem-Broker.

In Section 5, the resource type decision scheme and VM instance allocation method are introduced so as to provide cloud resource to service users efficiently.

5. Request Scheduling Scheme using MRSV

After getting the expected resource capability from user-specified SLA, to decide which type of VM resource should be allocated to the current sub-task, we adopt the scheme shown in Fig 7. in the VM Allocation Manager. The particular instance of the requested VM type to be allocated to the current sub-task will be selected by the scheme shown in Fig 8.

As discussed in section 2.3, we adopt the framework for workflow scheduling with bipartite SLA definitions which are SLA_1 and SLA_2 to give user the abstraction

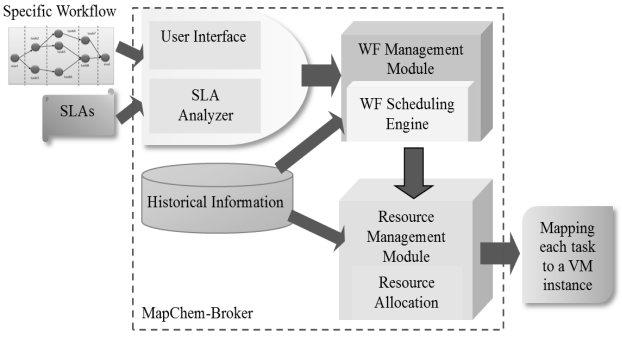


Fig. 6. Working procedures of MapChem-Broker

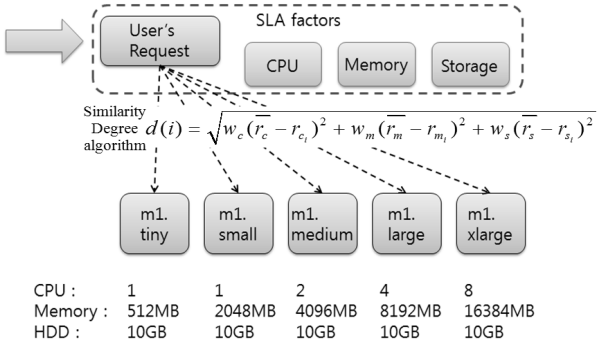


Fig. 7. Procedure of VM Type decision

from complexity of system information. SLA_1 is user-familiar service requirements which include deadline and total cost. and SLA_2 is systematic requirement like CPU, Memory, Storage, Network, etc. The Similarity Degree Function [16] is calculated as the Euclidean distance between two vectors each of which contains SLA_2 parameters. Since SLA_2 parameters reflect the resource capability, we can infer that a smaller similarity degree value indicates the more similar resource capability between the user-expected resource type and available resource type from an overview. So we will select the VM type that has the smallest Similarity Degree to process the being considered sub-task. VM type decision procedure starts from transforming the SLA_1 into SLA_2 to decide the resource specifications (parameters) to allocate optimized resource supply. To transform the SLA_1 to SLA_2, we need to define the SLA conversion function; however this is another complex problem so it will be studied in future works. So in this paper, we only focus on mapping the SLA_2 to a particular resource instance. There are several VM types which are prepared in advance. By using the Similarity Degree Function as mentioned earlier, users' requests can be allocated to the most proper VM type. By selecting the right VM type which has the most similar resource parameters with user-expected ones, we are very likely to get the expected performance from the VM, which is to comply with the resource parameters specified in SLA_2 as much as we can so that we can guarantee the QoS requirements specified in SLA_1.

INPUT :
$[\overline{r_c}, \overline{r_m}, \overline{r_s}]$: the user-required resource specification for executing cloud service.
$vt_i = [r_{c_i}, r_{m_i}, r_{s_i}]$: the resource specification for some type of VM.
c_i : the capacity of request numbers allocated to VM type i .
v_{ij} : the VM instance belonging to the VM type vt_i .
VARIABLES :
$d(i)$ is the similarity degree between the resource specification of the VM type of i and the required resource specification of cloud service user.
w_c, w_m, w_s represent the weighted values for the CPU, memory, and storage of a resource.
$wl(v_{ij})$ is the current workload in terms of the number of being processed job requests in VM instance v_{ij} , $0 \le wl(v_{ij}) \le c_i$ for each v_{ij} .
OUTPUT :
v_{sei} : the selection of a VM instance to execute the job request.
BEGIN :
Calculate the similarity degree using the formula referenced from [16].
And then find the minimum $d(i)$ among all the values of similarity degree to fix the VM type.
For selected VM type vt_i , find $v_{sei} = \arg\max(wl(v_{ij}))$, with the constraint that $wl(v_{ij}) < c_i$.
END

Fig. 8. Procedure for resource instance decision

After the resource type decision, to allocate user's request to a particular VM instance with maximization of VM instance utilization we adopt the MRSV scheme as discussed in section 3. When the VM type of a user's request is selected, the VM allocation module checks the current processing number of requests in each VM instance of the selected type. The upper limit threshold for allocation capacity of each VM instance is the value obtained from many empirical tests and that achieves the maximization of resource utilization. The rule for consideration in selecting VM instance is that we choose the instance that is with the most amount of requests being processed among all instances of the same type and also that the amount of the being processed requests is under the instance's upper limit of request amount, so that we utilize the resource capability to the fullest of each VM.

The description for the steps of resource type decision and VM instance selection is shown in Fig 8. in detail.

6. Experimental Results

We evaluate the proposed scheme in Section 5 in terms of performance, cost, and cost-performance. We compare the results of our proposed scheme under the Cloud testbed with that of an existing scheme, the "Dynamic Scaling Scheme (DSS)" scheme in [17], which is also under the cloud based environment. The different points between two schemes are shown in Table 2. The proposed scheme implements similarity degree algorithm and the MRSV scheme, while the existing DSS apply the random VM allocation and the SRSV scheme.

In this paper, to mimic the cloud resource provider, we establish the open-source Cloud platform called OpenStack and use the component named Nova in the

Table 2. Comparison of Proposed Scheme and DSS in cloud testbed

	Proposed Scheme	DSS
Different Point 1	VM provisioning with similarity degree scheme	VM provisioning without similarity degree scheme, instead, with the random VM type allocation
Different Point 2	One VM instance can process multi requests (MRSV)	One VM instance can only process one request (SRSV)

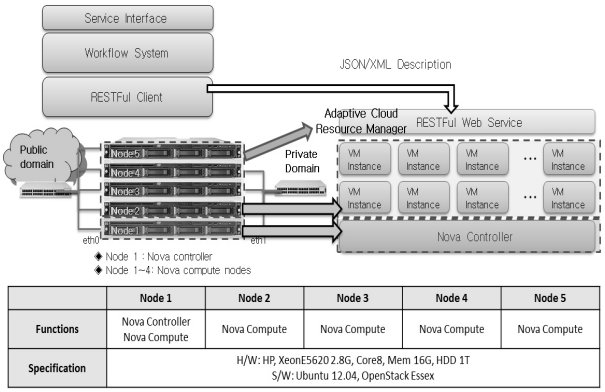


Fig. 9. Experimental environment [16, 18]

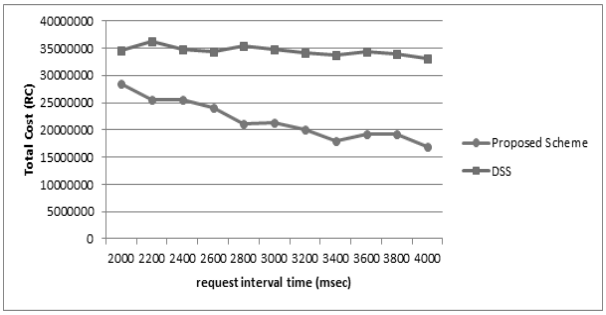


Fig. 10. Cost comparison of the Proposed Scheme and DSS with inter-arrival time variations of chem-app tasks

OpenStack which provides computing services to users and manages VM life cycle, as shown in Fig. 9. In OpenStack platform the operating front end machine is called Nova controller node. The controller node manages the operations between Nova computing nodes. The actual computing work is processed on Nova computing nodes and their status are reported to the Nova controller node. Requests from users are submitted to the system through the public IP address, and our system requests resources to the Nova controller node through the public/private IP address and the actual job request is send to the provided VM instances through the nova virtual network.

In the following experiments, the adopted application model is the chem-app task/job which is the computation of the QSAR table for various SDF_XX input molecular files. The workflow model is simplified by a DummyClient which sends QSAR computation jobs to the broker and we can program to set the requests' generating speed and generating period. We generate job requests within the same length of time period with different interval time. On receiving the job request, the MapChem-broker deploys its workflow scheduling module to map input job's SLA into expected resource parameters which will be used by the Similarity Degree Algorithm in the resource type decision process. The job requests for the same kind of resource type will go into the shared queue for that resource type. The jobs may spend some time waiting to start to get processed in the queue. By adopting the proposed resource allocation mechanism, a VM instance will be allocated for processing the job request. As for the resource capability parameters, we ignore the storage size because all types of job requests in experiments of this paper do not require the storage; only CPU and RAM are necessary for processing those job requests.

Fig 10. shows the resource allocation cost of both the proposed scheme and the DSS. The cost of proposed scheme is cheaper than DSS. As mentioned earlier, since the proposed scheme adopts MRSV, the utilization of VM instance is maximized by processing the multi requests as many as possible in each single resource. As a result, fewer VM instances will be needed so that the operation cost of resources can be reduced compared to DSS which adopts the SRSV scheme. Furthermore, as the workload of Mapchem application is decreased, the cost of MRSV is also decreased since the possibility of requests-grouping allocation is increased when the resources are not busy.

The performance of proposed scheme and DSS is defined as the inverse value of total completion time of all job requests, which is shown in Fig. 11. In our evaluation, surprisingly the performance of proposed scheme is better than the DSS in some cases. In this experiment, since the processing time of application tasks is considerably small as compared to the overhead of VM instance generation, the effect of the VM-generation delay of SRSV is bigger than the effect of separation of resource capacity and context switching delay of MRSV; therefore the performance of SRSV is not as good as compared to the proposed MRSV scheme, in some cases. However, in general case, the processing time of a

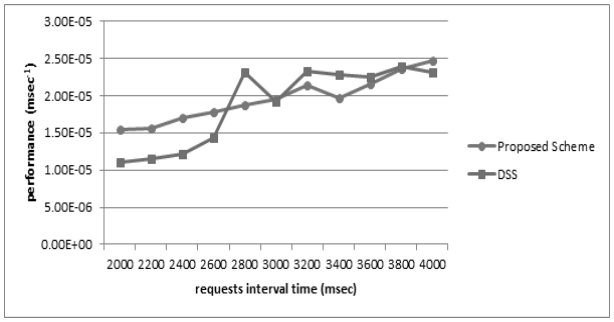


Fig. 11. Performance comparison of Proposed Scheme and DSS with inter-arrival time variations of chem-app tasks

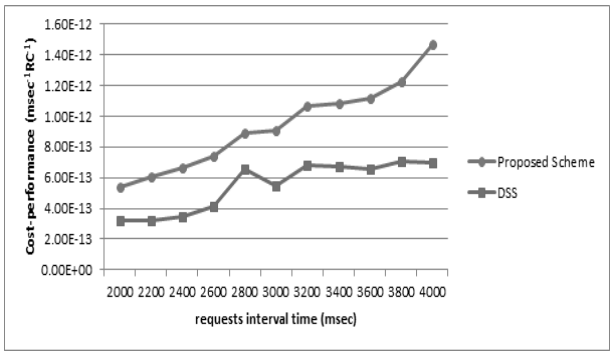


Fig. 12. Cost to Performance comparison of Proposed Scheme and DSS with inter-arrival time variations of chem-app tasks

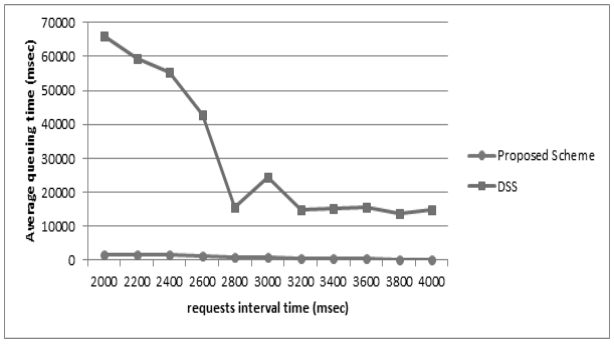


Fig. 13. Average queueing time comparison of Proposed Scheme and DSS with inter-arrival time variations of chem-app tasks

request is much longer than the generation time of a VM instance, so the result graph of performance of proposed scheme can be shown differently in practice.

Fig. 12 shows the cost to performance of two schemes. The cost to performance here is calculated as performance divided by cost (in terms of their numeric values) and it is an integrated metric to evaluate the cost-efficiency for the utilization of a computing resource. We see the cost-performance for the proposed scheme beats that of DSS. This proves that the proposed scheme is optimizer than DSS for VM provisioning. All this advantage owes to that the proposed scheme can process multi requests using one VM instance and also that the proposed scheme adopts the similarity degree scheme.

In Fig. 13, we can check the average queueing time of proposed scheme and DSS. The average waiting time of SRSV is significantly longer than the MRSV scheme in the entire workload range. This result is compatible with the result of Fig. 11, as mentioned above, since the VM-generation delay of a VM instance in SRSV prevails the effect of the performance degradation of context switching and other drawbacks in MRSV.

7. Conclusions

In this paper, we proposed the framework for scheduling workflow in the cloud with a new VM allocation scheme implemented in the broker in order to minimize the resource operation cost while guaranteeing the acceptable QoS. The proposed VM allocation scheme called Multi Requests to Single VM (MRSV) is the key feature in our system compared to traditional researches about cloud computing. In MRSV scheme, multi requests are able to be allocated to one VM instance within the acceptable performance, so the cost of resource operation can be reduced and even the performance can be improved in some special cases. In addition, the mapping from requests to VM instances can be performed efficiently by using the simple similar degree function in the resource type decision.

Through the experimental evaluation, we show that our proposed VM allocation scheme (with MRSV) outperforms the existing scheme (with SRSV) in views of cost, cost-performance and average queuing time. With all the discussions, we conclude that the proposed framework and VM allocation scheme can be a reasonable approach to improve the automation and cost-efficiency of cloud environment. In future work, we will upgrade the application model in these experiments from individual jobs to interdependent workflow jobs and define formal SLA description for the users and find solutions for translating users' SLA into proper resource specification parameters.

Reference

[1] F. Machida, M. Kawato, and Y. Maeno, "Just-in-Time Server Provisioning Using Virtual Machine Standby and Request Prediction," in *2008 International Conference on Autonomic Computing*, 2008, pp.163 - 171.

[2] A. Fox and R. Griffith, "Above the clouds: A Berkeley view of cloud computing," University of California, Berkeley, pp.7 - 13, 2009.

[3] M. Mao, J. Li, and M. Humphrey, "Cloud auto-scaling with deadline and budget constraints," *11th IEEE/ACM International Conference on Grid Computing*, pp.41 - 48, Oct., 2010.

[4] H. Topcuoglu, S. Hariri, and M.-Y. W. M.-Y. Wu, "Performance-effective and low-complexity task scheduling for heterogeneous computing," *IEEE Transactions on Parallel and Distributed Systems*, Vol.13, No.3, pp.260 - 274, 2002.

[5] M. Mao and M. Humphrey, "Auto-scaling to minimize cost and meet application deadlines in cloud workflows," *in 2011 International Conference for High Performance Computing Networking Storage and Analysis SC*, 2011, pp.1 - 12.

[6] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludascher, and S. Mock, "Kepler: an extensible system for design and execution of scientific workflows", in *Proceedings of the 16th International Conference on Scientific and Statistical Database Management*. Washington, DC, USA: IEEE Computer Society, 2004, pp.423-.

[7] M. H, S. E, S. H, and D. J. The triana project. [Online]. Available: http://www.trianacode.org/.

[8] L. Liu, C. Pu, and D. D. A. Ruiz, "A systematic approach to flexible specification, composition, and restructuring of workflow activities", *J. Database Manag.*, Vol.15, No.1, pp.1-40, 2004.

[9] E. Deelman, J. Blythe, Y. Gil, and C. Kesselman, "Grid resource management", J. Nabrzyski, J. M. Schopf, and J. Weglarz, Eds. Norwell, MA, USA: Kluwer Academic Publishers, 2004, ch. Workflow Management in GriphyN, pp.99-116.

[10] R. Armstrong, D. Hensgen, and T. Kidd, "The relative performance of various mapping algorithms is independent of sizable variances in runtime predictions", in Proceedings of *the 7th Heterogeneous Computing Workshop*, ser. HCW'98. Washington, DC. USA: IEEE Computer Society, 1998, pp.79-.

[11] F. Magoules, T.-M.-H. Nguyen, and L. Yu, *Grid Resource Management: Towards Virtual and Services Compliant Grid Computing*. Boca Raton, FL, USA: CRC Press, Inc., 2008.

[12] A. Silberschatz, P. B. Galvin, and G. Gagne, *Operating System Concepts*, Vol.32, No.8. Wiley, 2005, pp.921.

[13] J. C. Browne, J. Lan, and F. Baskett, "The interaction of multi-programming job scheduling and CPU scheduling," *Proceedings of the December 5-7, 1972, fall joint computer conference, part I on - AFIPS '72* (Fall, part I), pp.13, 1972.

[14] P. Denning, "Thrashing: Its causes and prevention," *Proceedings of the December 9-11, 1968*, fall joint, 1968.

[15] R. Reddy and P. Petrov, "Cache partitioning for energy-efficient and interference-free embedded multitasking," *ACM Transactions on Embedded Computing Systems*, Vol.9, No.3, pp.1 - 35, Feb., 2010.

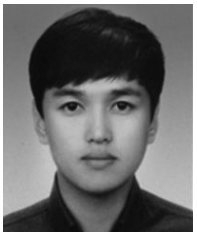
[16] S.-H. Kim, D.-K. Kang, Y. Ren, Y.-S. Park, K.-N. Joo, C.-H. Youn, Y. S. Park. "An Experimental Cloud Resource Broker System for Virtual Application Control with VM Allocation Scheme", to appear in *the 7th International Conference on Ubiquitous Information Technologies & Applications (CUTE)*, Hong Kong, Dec., 2012.

[17] T. C. Chieu, A. Mohindra, A. A. Karve and A. Segal. "Dynamic Scaling of Web Applications in a Virtualized Cloud Computing Environment". *IEEE International Conference on e-Business Engineering*, 2009.

[18] D.-K. Kang, S.-H. Kim, Y. Ren, B.-S. Kim, W.-J. Kim, Y.-S. Kim, C.-H. Youn, C. S. Jeong. "Enhancing a Strategy of Virtualized Resource Assignment in Adaptive Resource Cloud Framework", to appear in I*nternational Conference on Ubiquitous Information Management and Communication (ACM ICUIMC)*, Malaysia, Jan., 2013.

Ye Ren

She received the B.S. degree in Optoelectronic Information Engineering from Harbin Institute of Technology, Harbin, China in 2010, and the M.S. degree in Electrical Engineering from Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea in 2012. Currently she is a researcher in Grid Middleware Research Center in KAIST. Her research is in the field of workflow management in distributed computing environment such as grid or cloud computing. Email: catherine@kaist.ac.kr



김 성 환

He received the B.S. degree in Media and Communications Engineering from Hanyang University, Seoul, Korea in 2012. Now he is a M.S. candidate in Department of Electrical Engineering at Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea and is member of Advanced Network and Computing Laboratory in KAIST. His research interests include mobile cloud computing and cloud collaboration.

강 동 기

He received a M.S degree in Department of Computer Engineering from Chonbuk National Univ, Jeonju, Korea, in 2011. He is currently working toward the Ph.D. degree in Electrical Engineering at Korea Advanced Institute of Technology (KAIST), Daejeon, Korea. His main research interests include Virtualized Resource Management, and Cloud Computing.

김 병 상

Byungsang Kim received an MS degree from KAIST in 2004 in information and communications engineering and a BS degree from Dongkuk University in 2002. He was a researcher in Korea e-Science research group in KISTI, Korea from 2006 to 2009. He is currently a Ph.D student in the Dept. of information and communications engineering in KAIST, Korea. His research is in the field of resource management with scalability in elastic computing environments such as grid or cloud computing and the large-scale data analysis platform and big data analytics.

윤 찬 현

Chan-Hyun Youn received the B.Sc and M.Sc degrees in Electronics Engineering from Kyungpook National University, Daegu, Korea, in 1981 and 1985, respectively. He also received a Ph.D. in Electrical and Communications Engineering from Tohoku University, Japan, in 1994. Since 2009, he has been a professor at Department of Electrical Engineering in KAIST, Daejeon, Korea. He also was a Dean of Office of Planning Affairs and a Director of Research and Industrial Cooperation Group at former Information and Communications University, in 2006 and 2007. He was a Visiting Professor at MIT in 2003 and has been engaged in the development of Physio-Grid system with Prof. R.G. Mark's Group in LCP (Laboratory for Computational Physiology) of MIT since 2002. He also is a Director of Grid Middleware Research Center at KAIST. Where, he is developing core technologies that are in the areas of mobile cloud, mobile collaboration system, Internet computing workflow management, distributed network architecture, communication middleware, advanced e-Healthcare system, e-Health application services and others. Currently, he is serving the Editor-in-Chief of KIPS (Korea Information Processing Society), and an Editor of Journal of Healthcare Engineering (U.K.), and served head of Korea branch (computer section) of IEICE, Japan (2009, 2010). He is a member of IEEE, KICS and IEICE, respectively.