Towards Automated User-Centric Cloud Provisioning: Job Provisioning and Scheduling on Heterogeneous Virtual Machines

1) Eman Hossny                            Sara Salem                         Sherif Khattab  
2) Computer Science Department  
3) Faculty of Computers and Information, Cairo University  
4) Cairo, Egypt  
{e.hossny, s.salem, s.khattab@fci-cu.edu.eg}

Abstract - The cloud Infrastructure-as-a-Service (IaaS) model, which provides users with remote virtual machines in a pay-per-use method, is attracting more people to start using IaaS services. This work advances the state-of-the-art one step further towards the realization of an automatic user-centric cloud provisioning and job scheduling tool. First, we extend some of the existing provisioning and scheduling strategies to be applicable to a real infrastructure of heterogeneous virtual machines. Second, we evaluate the extended strategies using the MetaCentrum data set. Our results show that handling different types of machines and applying backfilling scheduling successfully increased the average machine utilization and decreased the average job waiting time.

Keywords - cloud computing; job scheduling; virtual machine provisioning

I. INTRODUCTION

The cloud computing model aims at providing virtualized resources to be used and shared by users who access these resources on demand and pay only for their actual use. This computing model fostered a number of business models that come with advantages to both small and large enterprises. For small enterprises, the initial startup capital is minimized. In addition, the maintenance and management overhead of the resources is carried by the provider. Even large enterprises can make benefits from the cloud by extending their IT resources in case of maximum workload to serve more customers [1].

Cloud computing services are commonly offered at three layers of abstraction: Software-as-a-Service (SaaS), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS). Our work focuses on the IaaS layer. IaaS provides virtual machines (VMs) with specific capabilities, such as number of CPUs, memory, disks, networking and software packages. The users can provision and de-provision VMs according to their needs [2]. Amazon EC2 [3] and GoGrid [4] are examples of IaaS providers. They both offer on-demand and reserved pricing models. In the on-demand model, the user pays for the number of used hours, that is, the number of hours the rented VM was up. In the reserved model, a VM is reserved for a long period (1-3 years) with certain reservation fees, and additionally an hourly rate (though lower than on-demand rate) is applied. Amazon also offers the spot pricing model, which allows users to bid with hourly rates on unused machines. These machines will be allocated to the user as long as the user's bid is larger than the machine's floating price.

Provisioning a virtual machine is done by providing the user with access to a virtual machine with specific software and hardware capabilities to fulfill the user needs. Cloud providers usually automate provisioning by defining a set of virtual machine templates a priori and letting the user selects from this set. For example, Amazon offers six types of instances [3], which cover diverse computing requirements (e.g., high memory, high processing, and high parallelization). Although provisioning is automated, the user is still responsible to decide on the number of required machines, when to start and stop these machines, and scheduling the jobs on the machines (i.e., which machine will execute which job and when, if the jobs will run sequentially or in parallel and so on).

Research has recently targeted the automation of provisioning and scheduling operations in the cloud. The reason is that manual provisioning and job scheduling can lead to non-optimal results in terms of
waiting time, renting cost, and machine utilization. In addition, the manual operation can be very complex in the case of heterogeneous virtual machine requirements and large numbers of jobs.

The work in [5] presents a simulation study that aims at automating the provisioning and scheduling operations and examines a set of provisioning strategies with the objective of minimizing both the waiting time and the total cost. However, the assumptions made in [5] may limit the applicability of the results to real-life scenarios. First, the VMs are assumed to be identical. Second, an unlimited pool of VMs is assumed. Third, each VM executes one job at a time (a single processor per VM is implicitly assumed).

In our work, we relax these assumptions and extend the proposed provisioning strategies in [5] to address more realistic assumptions about the VMs and the jobs. We assume that jobs have different requirements (e.g., in terms of number of processors, required memory, installed software, network bandwidth, etc.) and that the VMs are heterogeneous in their computing and network resources. Although the cloud is usually said to offer unlimited resources and the resources can be practically huge, the available VMs still have a limit, especially in private clouds. Some VMs have multiple processors and thus can execute more than one job at a time. Like [5], we apply the on-demand billing model, whereby the billing unit is one hour.

The major contributions of our paper are as follows:

We extend state-of-the-art provisioning strategies to handle the provisioning of heterogeneous VMs to jobs with different resource requirements.

We investigate by simulation the effect of intra-VM job scheduling strategies. In particular, a VM can execute multiple jobs at the same time. Backfilling [6] is used to increase machine utilization, to overcome the restriction of having limited available VMs, and to decrease the waiting time of jobs.

Using simulation on a real dataset of job execution trace, we compared backfilling with FCFS. Backfilling scheduling proved to increase machine utilization and reduce both job waiting-time and total renting cost.

The rest of the paper is organized as follows. Section 2 provides a brief overview about basic VM provisioning and job scheduling strategies, which serve as the basis for our extensions. Section 3 explains the problem we address in this paper. Some of the related work is presented in section 4. Section 5 describes architecture for an automated cloud provisioning tool and describes our proposed extensions. Section 6 provides a simulation-based evaluation of our extended VM provisioning strategies. Finally, the conclusion and future work are presented in section 7.

II. BACKGROUND

This section describes briefly two well-known job scheduling strategies, namely FCFS and backfilling. This work investigates the effect of applying backfilling to job scheduling inside VM queues and compares the performance of backfilling to that of FCFS in terms of VM utilization, job waiting time, and total cost. In addition, this section describes the provisioning strategies studied in [5], which we extend in our work.

A. Job scheduling strategies

FCFS Strategy: In the First Come First Served (FCFS) strategy, the jobs run in the same order of their arrival. It is the simplest scheduling strategy. However, it increases the waiting time of jobs when the long-running jobs arrive before the short-running jobs. Also, this strategy decreases the utilization of multi-processor VMs, whereby the VMs may have some idle resources while some jobs still wait in the global job queue [6]. This situation happens when the available idle resources do not meet the requirements of the job at the queue head but meet the requirements of other jobs back in the queue.

Backfilling Strategy: In the backfilling strategy, a late-arriving job can run before (leap ahead of) an earlier-arriving job if its running does not delay the finish times of earlier job(s) in the queue. This strategy increases the utilization of the VMs by using idle resources that are not sufficient for the head of the queue but are enough for other jobs down the queue [6].

Backfilling algorithms can be categorized into three categories: aggressive, conservative, and flexible [7]. In the aggressive backfilling algorithm, only the job at the head of the queue is guaranteed not to be affected by the leaping jobs. In the conservative algorithm, each waiting job in the queue gets a reservation that cannot be violated by other leaping jobs. Conservative backfilling tends to achieve reduced performance compared to aggressive backfilling [7]. The Flexible backfilling algorithm decides whether to guarantee reservation for a job or not based on need (e.g., the waiting time of the job). We apply aggressive backfilling algorithm in our scheduler.

B. Virtual machine provisioning strategies

This subsection presents a brief description of the provisioning strategies studied in [5]. The 1VM4All strategy provisions only one VM to run all jobs. For identical VMs and jobs that have the same resource requirements (as in [5]), 1VM4All is the cheapest provisioning strategy. However, it suffers from long job waiting-time.

The 1VMPerJob strategy deploys a new VM for each job. This may be very costly; therefore, the authors in [5] improved the 1VMPerJob to 1VMPerJobPlus, 1VMPerJobBest, and 1VMPerJobWorst. 1VMPerJobPlus, instead of deploying a new VM, reuses a previously provisioned
VM that is both idle and still up. In 1VMPerJobBest, the selected VM is the one whose current billing unit (hour) will end the latest among the idle VMs, and in 1VMPerJobWorst the VM that will shutdown the soonest is selected. The family of 1VMPerJob strategies guarantee zero waiting time with an unlimited pool of VMs. However, with a limited number of VMs, jobs have to wait until a free VM becomes available.

The main idea of bin packing strategies, such as First Fit, Best Fit, Worst Fit, and Earliest Fit, is to make the jobs wait some time in the job queue of one of the active VMs as long as running this job will not extend the active time of the VM into a new billing unit (i.e., a new hour). In other words, these strategies aim at running the arriving job with no extra cost. Otherwise, a new VM is provisioned for the job. These strategies may increase job waiting-time, but they decrease the total cost.

Relax-based strategies specify a limit on the waiting time of jobs. A job can wait on a VM queue if its waiting time will not exceed its running time multiplied by a relax factor. RelaxFirst strategy selects any of the eligible VMs, RelaxEarliest selects one of the eligible VMs with the minimum idle time before end of hour, and the RelaxLatest, selects the maximum idle time VM. Relax-based strategies converge to 1VMPerJobPlus when the relax factor is small (e.g., 0.5) and to bin packing strategies when the relax factor is large (e.g., 10) [5].

III. PROBLEM DEFINITION

This work addresses two interweaved subproblems. The first is the provisioning of jobs with different resource requirements to a limited set of heterogeneous VMs, and the second is the scheduling of jobs assigned to each VM. The goal of the overall problem is to strike a user-controlled balance between the cost of VM rental, using the on-demand billing model with different hourly rate per VM depending on its capabilities, and the waiting-time of jobs. We assume that the user jobs are independent and that there is no preemption for jobs (i.e., a job cannot exit from its VM until its execution is terminated or failed).

To address this problem, we extend the strategies of [5] in two ways. First, we replace the FCFS scheduling strategy with backfilling on the newly arrived jobs inside each VM’s queue to increase VM utilization and minimize the waiting time of jobs. Second, we modified the VM provisioning strategies to work on limited number of heterogeneous VMs (i.e., VMs with different capabilities which lead to different cost per VM). Fig. 1 depicts an illustrating example using 1VMPerJobPlus and bin packing strategies. The depicted workload consists of four jobs: Job1-4. Jobs 1-3 can run on the same VM because they require the same software packages and number of CPUs but different memory sizes. One can deploy a new VM for each of these jobs, but this would generally leads to high cost with unlimited pool of VMs or long waiting-time in a system with limited number of VMs. 1VMPerJobPlus can provide better solution by using the idle resources of the active VM instead of starting a new VM. More specifically, in Fig. 1, when Job1 arrives at time t1, a new VM with minimum cost is deployed such that it meets the resource requirements of Job1. When Job2 arrives at t2, VM1 is active and has enough idle resources to run Job2. Therefore, Job2 runs in parallel with Job1 on the same VM. Job3 also arrives at t2, but currently VM1 does not have available CPUs to fulfill Job3 requirements. Therefore, a new VM is deployed for Job3. Since Job4 requires different
capabilities that none of the active VMs have, a new VM is deployed for Job4.

In the bin packing strategies, Job1 and Job2 run on VM1 and Job3 waits in the job queue of VM1 because running Job3 does not cause VM1 to enter in a new hour. After Job2 ends its execution, still the available memory in VM1 is not sufficient to run Job3. Therefore, Job3 waits until Job1 ends its execution. A new VM is deployed for Job4 because its requirements are not met by VM1. The bin packing strategies may increase the waiting time of some jobs, but they decrease the total cost.

IV. RELATED WORK

Job provisioning and scheduling on the cloud are challenging research problems that have motivated many researchers to make the best use of cloud resources with reduced cost. Researchers in [8] studied the benefits of applying the hybrid cloud concept to provide fast response time to customer requests with optimized cost. They analyze the impact of using different alternatives of backfilling algorithms to schedule the jobs on the VMs. They allow the jobs to have different requirements by specifying certain VM image to each job; in their experiments, however, they used only one type of Amazon EC2 instances. In our work we do not consider hybrid clouds, and we are more flexible in deciding which VM image can serve the job. Any VM that has capabilities that are equal to or exceeding the job requirements can run the job if it passes the eligibility and optimality checks. The work in [9] proposes an automated virtual machine provisioning algorithm that augments the local VMs with public ones from Amazon EC2 spot instances. The bidding is done by the rate of the on-demand instances. Spot instances are selected to reduce the total cost because their hourly rate is usually lower than the on-demand rate.

The authors in [10] proposed to reduce the cost of running a data-intensive application on the cloud by examining different data storage and communication models over Amazon S3. Also, they executed their application with different levels of parallelization by using different numbers of processors. Their simulation clarified that choosing the right provisioning strategy for storage and computing resources could reduce the cost without changing the performance. In our work, we model computation resources only and consider a complete VM as the provisioning unit not just a processor.

Another work which depends on the on-demand billing model of Amazon EC2 virtual machines is presented in [5]. The authors employed simple provisioning strategies that vary from very cheap strategies with large waiting time to very expensive strategies with zero waiting time. Their objective is to find a tradeoff between cost and waiting time for running a set of independent and non-preemptive jobs with known execution duration. They assumed that the virtual machines are unlimited and identical. The waiting jobs on a virtual machine’s queue are executed in FCFS. The work in [5] is the closest to ours with two main differences: 1) they assumed infinite number of VMs but this is not realistic. So, in our work, we assumed limited number of VMs. 2) they assumed that all VMs are identical. However, in a real system, the small jobs should require a small VM with limited capabilities and vice versa. Therefore, in our work, we give the ability to provision heterogeneous VMs.

The authors in [11] and [12] have a similar objective, where they proposed a dynamic VM provisioning mechanism to support both grid and cloud. They provided users with more satisfaction by serving their requests in a fast responsive way. They enhanced the resource utilization by using online clustering approach to analyze the arrived jobs and know the types of required VMs. Then the information coming from the analysis is used to provision the VMs. While in our work, we improved the resource utilization by applying backfilling and allocating the VMs according to the job requirements.

There are other papers that address the VM provisioning problem on the server side, where their objective is to maximize the utilization of the infrastructure as a whole. Inomata et al. [13] proposed a dynamic resource allocation method to add and delete the VMs based on their load on IaaS. However, our work addresses the VM provisioning problem on the client side.

V. AUTOMATIC CLOUD PROVISIONING ON HETEROGENEOUS VMs

In this section, we describe an architecture for a user-centric tool to provision and schedule jobs on the cloud. The architecture is applicable on a real IaaS to schedule user jobs under user control. The target is to minimize the total cost and the jobs’ waiting time, in addition to increasing the utilization of the virtual machines. In this architecture, we take care of heterogeneity in the user jobs and the VMs because most—if not all—cloud providers offer different types of VMs that are not identical.

A. Tool architecture

Fig. 2 depicts the main components of the proposed automatic provisioning tool. It is composed of four components: the job scheduler, the VM activation component, the shutdown manager, and the statistical component. Also, the architecture assumes a limited number of VMs that are divided into two pools: active and non-active VMs.

The job scheduler component receives a job with its requirements in terms of software requirements, main memory size, number of CPUs, and number of nodes (VMs) that the user needs to run his job. The job scheduler component uses aggressive backfilling strategy and one of our extended VM provisioning strategies to assign a VM for the newly arrived job. The active VMs are first checked for eligibility to handle the job. An active VM can run
multiple jobs at the same time as long as the resources are sufficient for the running jobs. If none of the active VMs is eligible, the non-active VMs are checked for eligibility. The optimum one, which has the minimum cost, of the eligible VMs is selected and sent to the VM activation component. It happens in some cases that all non-active VMs are not sufficient to execute the job. In this case the job waits on one of the active VMs that is optimum to satisfy the job requirements. Algorithm 1 purifies the process of job scheduler component. The logic of the “eligible_sufficient” and “optimum” functions differs based on their provisioning strategy.

The VM activation component activates the optimum VM and moves it from the non-active VMs to the active VMs pool. Its process is clarified in algorithm 1. The shutdown manager component ensures that, according to the Service Level Agreement (SLA), the active VMs are shutdown after certain amount of time – if they are idle and about to finish their billing unit- and are moved back to the available non-active VMs pool.

The statistics component collects statistics at two events. At the job termination time, this component calculates the waiting time of the job (Algorithm 2). Also, at the VM shutdown time, it calculates the utilization of the VM and the cost of using it.

In our work, each virtual machine has the following properties: id, number of processors, processor speed, memory size and supported software packages. The price of the virtual machine is proportional to its capabilities. Equations 1 to 4 illustrate how the virtual machine price is calculated.

The total cost for using N number of VMs to execute set of jobs is illustrated in equation 5.

\[
\text{Price(VM)} = \text{Price(Processors)} + \text{Price(Main Memory)} + \text{Price(Software)}
\]

\[
\text{Price (Processors)} = \text{No. Processors} \times \text{Processor Speed} \times \text{PROC\_UNIT\_PRICE}
\]

\[
\text{Price (Main Memory)} = \text{Main Memory Size (KB)} \times \text{MEM\_UNIT\_PRICE}
\]

\[
\text{Price (Software)} = \text{No. Software packages} \times \text{SOFT\_UNIT\_PRICE}
\]

\[
\text{Total Cost} = \sum_{i=1}^{N} \text{Active Hours(VM}_i\} \times \text{Price (VM}_i\}
\]

B. Extended VM provisioning and scheduling strategies

The main differences between the provisioning strategies in [5] and the extended strategies in this paper are: 1) the backfilling is applied on the waiting job queues of the VMs. 2) as the VMs are limited, we do not deploy new VM when there is no one available, but allow jobs to wait until one VM becomes available. The 1VM4All strategy is considered not applicable for the heterogeneous nature of our system. It’s very difficult to find one VM that can satisfy the requirements of all the jobs,
and that’s what happened in the MetaCentrum data set [14] used in our evaluation. The software requirements of all the jobs were not found together on 1 VM.

As described in Section 2, one can use the backfilling or FCFS to schedule the waiting jobs to run on VMs. The aim of FCFS is to provide fairness between the waiting jobs. However, the backfilling tries to utilize the available resources. We applied the backfilling and FCFS on the waiting job queue of VMs.

Algorithm 1: The process of job scheduler component and Activation component

Inputs: List<VM> non-activeVMs (size), List<VM> activeVMs
Output: VM targetVM
List<VM> candidateVMs = eligible_suffient(j, activeVMs);
IF candidateVMs.size() = 0 THEN
    candidateVMs = eligible_suffient( j , non-activeVMs);
END IF
IF candidateVMs.size() = 0 THEN
    candidateVMs= eligible_wait ( j , activeVMs);
END IF
IF candidateVMs.size() > 0 THEN
    end = optimum(candidateVMs);
ELSE
    targetVM = null;
END IF
IF targetVM = null THEN
    lostJobsCount++;/*count jobs that are not sufficient for any VM*/
ELSE
    IF notActive(targetVM) THEN
        activate(targetVM);
        assign(j, targetVM);
    END IF
    IF sufficient_run(targetVM, j) THEN
        targetVM.currentRunningJobs.add(j);
    END IF
    IF targetVM.waitingJobs.size() > 0 THEN
        topJob = targetVM.waitingJobs.get(0);
        IF sufficient_run(targetVM, topJob) THEN
            targetVM.waitingJobs.remove(topJob);
            targetVM.provision(topJob);
        END IF
        IF targetVM.currentRunningJobs.size() > 0 THEN
            topJob = targetVM.currentRunningJobs.get(0);
            IF sufficient_run(targetVM, topJob) THEN
                targetVM.currentRunningJobs.remove(topJob);
                targetVM.provision(topJob);
            END IF
        END IF
    END IF
END IF

Algorithm 2: The process of job departure

Inputs: Job j
Output: waitingTime of j and start the next job (if possible)
calculateWaitingTime(j);
targetVM=getAssignedVM(j);
/*release the resources from the job j*/
targetVM.de-provision();
IF targetVM.waitingJobs.size() > 0 THEN
    topJob = targetVM.waitingJobs.get(0);
    IF sufficient_run(targetVM, topJob) THEN
        targetVM.waitingJobs.remove(topJob);
        targetVM.provision(topJob);
    END IF
    IF targetVM.currentRunningJobs.size() > 0 THEN
        topJob = targetVM.currentRunningJobs.get(0);
        IF sufficient_run(targetVM, topJob) THEN
            targetVM.currentRunningJobs.remove(topJob);
            targetVM.provision(topJob);
        END IF
    END IF
END IF

VI. EVALUATION

In this section we present our simulation results of the extended provisioning and scheduling strategies. We first describe the used dataset, and then we summarize and briefly discuss the results.

A. Data Set

In order to evaluate our work, we used a real dataset, which includes a set of heterogeneous machines and a set of jobs with specific requirements. The data set used by [5] is relatively simple and does not provide such level of complexity. This is why we used the MetaCentrum dataset [14, 15]. MetaCentrum was previously used for complex job scheduling simulations [16]. It collects data for 103,656 jobs for 147 users executed on 14 heterogeneous clusters with different hardware setup (number of CPUs, memory size, and CPU speed), operating systems, and software packages.

In the published data set [14], there were a number of jobs that ran on cluster_10, which seemed to require memory that is more than the memory of the cluster machines. We consulted the authors and they indicated that some machines within cluster_10 had 7 GB while others have only 4GB. Moreover, there was some fluctuation in time as machines were upgraded. They suggested a fix by setting all machines in that cluster to have 7GB of memory. We adopted the fix and, additionally, we excluded any job that still required capabilities that are not available on any node (69 jobs ignored). Also, all jobs that required processors that were distributed on multiple nodes were excluded from our evaluation; in future work we can tackle these jobs too. So, the total number of used jobs in our simulations was slightly reduced to 100,375 jobs.

B. Results and Discussion

In our study, with more realistic job and VM assumptions than assumed in [5], backfilling scheduling proved to increase the machine utilization and reduce both job waiting-time and total renting cost (Figure 3, Figure 4, and Figure 5). In backfilling, the four variants of 1VMPerJob strategies produced better results than bin packing and relax-based strategies. They produced the minimum waiting time. In addition, 1VMPerJobPlus resulted in the minimum cost with maximum utilization as well.

VII. CONCLUSION AND FUTURE WORK

In this paper, we have studied the extension of provisioning and scheduling strategies to handle jobs with different resource requirements on heterogeneous virtual machines on the cloud. We extended a set of VM provisioning strategies to be applicable to a real infrastructure of a finite pool of heterogeneous virtual machines and jobs with different capabilities. Since we have a limited number of heterogeneous VMs, the backfilling scheduling strategy is applied on the job queue of VMs to increase their utilization. Using simulation on a real dataset of job execution trace, we compared
backfilling with FCFS. Backfilling scheduling proved to increase machine utilization and reduce both job waiting-time and total renting cost.

For simplicity, we deal with the jobs that need to run on only one node. In the future we will take care of running jobs on multiple nodes, job dependency, and prioritization of jobs. In this paper, we used only one billing model, which is the on-demand model. Future work may investigate different billing models so that the provisioning tool allows the user to select the best model.

References

Simulation Tools and Techniques (ICST), Barcelona, Spain, pp. 61-71, 2010.