Scheduling Jobs in the Cloud Using On-demand and Reserved Instances

Siqi Shen[1], Kefeng Deng[1,2], Alexandru Iosup[1], and Dick Epema[1]

[1] Delft University of Technology, Delft, the Netherlands
{S.Shen,A.Iosup,D.H.J.Epema}@tudelft.nl
[2] National University of Defense Technology, Changsha, China
Dengkefeng@nudt.edu.cn

Abstract. Deploying applications in leased cloud infrastructure is increasingly considered by a variety of business and service integrators. However, the challenge of selecting the leasing strategy — larger or faster instances? on-demand or reserved instances? etc.— and to configure the leasing strategy with appropriate scheduling policies is still daunting for the (potential) cloud user. In this work, we investigate leasing strategies and their policies from a broker's perspective. We propose, *CoH*, a family of Cloud-based, online, Hybrid scheduling policies that minimizes rental cost by making use of both on-demand and reserved instances. We formulate the resource provisioning and job allocation policies as Integer Programming problems. As the policies need to be executed online, we limit the time to explore the optimal solution of the integer program, and compare the obtained solution with various heuristics-based policies; then automatically pick the best one. We show, via simulation and using multiple real-world traces, that the hybrid leasing policy can obtain significantly lower cost than typical heuristics-based policies.

1 Introduction

A growing number of applications are running in the cloud. Academia [1, 2, 6, 16, 17, 23, 29] and industry [30] are both increasingly using cloud resources as infrastructure to serve their users, due to the elastic, flexible, and pay-as-you-go features of Infrastructure-as-a-Service (IaaS) clouds. Cloud brokers need to lease resources from IaaS clouds cheaply, yet execute the users' jobs in time. To achieve this, cloud brokers must use scheduling policies that match diverse requirements. *Finding scheduling polices that can schedule diverse workloads with zero waiting time yet cheaply is the focus of this work*

IaaS clouds offer their users various types of machine configurations: different amount of CPU cores, memory, and disk. It is non-trivial for a cloud broker to determine the combination of machine configurations for user demands. This situation is complicated by the current IaaS pricing models: machines configurations are not priced linearly with their performance. For example, an EC2 `large` instance can serve more web requests per core than the `small` instance, but their price per core is the same [24]. Moreover, for the same machine configuration, the clouds offer different billing options on-demand-, reserved-, and spot-instances, which are charged differently. Scheduling enough resources to meet user demands yet keep the cost low while adapting to workload changes remains challenging, despite recent research efforts [19, 24, 28].

In this work, we present a Cloud-based, online, Hybrid scheduling policy (*CoH*), which keeps the rental cost of cloud resources low by finding the best combination of machine configurations and billing options. At the core of this policy are its provisioning and allocation strategies. We formulate these strategies as Integer Programming Problems (IPP). As *CoH* needs to be executed online, the time to obtain a decision should be low. We limit the time to solve the IPP, and run simultaneously various heuristics. The *CoH* compares the result of IPP and heuristics, and picks the best one as its scheduling decision. Thus, a novel aspect of *CoH* is its portfolio-based scheduling strategy [12] adapted to IaaS clouds. Further, we devise, *CoH-R*, an extension of *CoH* to also makes use of reserved instances, which can lead to significant cost reduction compare to policies that use on-demand instances only.

The major contributions of this work are three-fold.

1. A novel online scheduling policy, *CoH*, which makes scheduling decision using a portfolio of IPP and heuristics-based approaches (Section 3).
2. A policy extended from *CoH*, *CoH-R*, which also makes use of reserved instances to reduce rental cost (Section 4).
3. An evaluation of our policies for two broad application domains, grid computing and online game hosting, using trace-based simulation (Section 5).

2 System model

2.1 Workload and Resource Model

The workload model in this work is a set of independent jobs. The resource requirements and the runtime of each job are known when the job arrives in the system. Once started, jobs run to completion, so we do not consider task preemption or migration during execution.

Each job can be described by a tuple (r_i, a_i, d_i), where r_i is the resource requirement of job i, a_i is the arrival time of job i, and d_i is its departure time. We assume that a computer can host one or multiple jobs. This model is similar to the work of Stillwell et al [26]. This kind of jobs is common: a compute node can run multiple MapReduce tasks; an online game hoster may consolidate several game servers on the same machine; etc. The resource requirements of each job, r_i, could be a vector indicating multiple resource requirements (e.g., CPU and Memory), or a scalar value (e.g., CPU only). We focus on the CPU requirement. In practice, r_i can be obtained though profiling [14, 24] or can be provided by the user.

We model the operation and billing model of cloud providers based on the real case of Amazon EC2. We assume that clouds have infinite capacity. Each newly provisioned VM needs serval minutes to be booted [14, 19]. An VM is charged per hour; even a factional consumption of less than one hour is counted as one hour. An VM indexed by j, has capacity denoted by w_j and hourly cost c_j.

2.2 Scheduling model

In our scheduling model, all machines are provisioned exclusively from clouds. The cloud broker has pre-configured and stored in the cloud all the necessary VM images

to run users' jobs. All the incoming jobs are enqueued into a queue. A system-level scheduler, running on a dedicated system, manages all the jobs and a pool of machines, and decides whether to provision new VM from clouds and/or to allocate jobs to VMs.

The scheduler is executed periodically (e.g., every 10 seconds). At each scheduling moment, the scheduler performs five tasks: (1) Predicting future incoming workloads; (2) Provisioning necessary VMs in advance, from clouds; (3) Allocating jobs to VMs, (4) Releasing idle VMs (which don't have job running on them) if its Billing Time Unit (BTU) is close to increase (e.g., 10 second before the leased hour). (5) If the wait time of un-allocated jobs is high, starting the necessary number of VMs. We design in the next section a scheduling policy, *CoH* to perform tasks (3) and (4). We further extend this policy in Section 4 to also use reserved cloud instances. As workload prediction is not the focus of this paper, we assume that there exists a predictor that can achieve perfect prediction of future workload. Relatively good predictor [18] already exists for the type of workload we target in this work.

3 Scheduling using on-demand instances

This section describes *CoH*, a Cloud-based, online, Hybrid scheduling policy using on-demand instances. The strategy of *CoH* is presented in Section 3.1. *CoH* needs to take both provisioning and allocation decisions, that is, to find a combination of VMs, and a mapping between jobs and VMs. We formulate the above problem as an Integer Programming Problem (IPP) in Section 3.2 and then select various heuristics to assist *CoH* in Section 3.3.

3.1 Policy Overview

CoH actively provisions VMs before they are needed, and maps jobs to already provisioned VMs according to the best mapping it can find. *CoH* finds the combination of VMs and the mapping by solving an online scheduling problem through solving (partially) one IPP and by using several heuristics, independently and simultaneously. As an online scheduler, *CoH* needs to take scheduling decisions within limited amounts of time; thus, it limits the time used to solve IPP, and compares the result of the IPP and heuristics. *CoH* acts as a portfolio-based scheduler, in which multiple strategies are considered simultaneously at each scheduling moment. The strategy that has the best objective value (defined in formula (1)) is picked as the scheduling decision. Heuristics are needed because the solution of IPP under limited time may be suboptimal or even infeasible (*CoH* may not find a feasible solution of the IPP in limited time).

3.2 Formalization of the Scheduling Problem

CoH needs to provision enough number of VMs to support all the incoming jobs, and to allocate all the jobs smartly such that the rental cost is minimized. An VM that has jobs running on it cannot be shut down, so it will still incur cost. Unnecessary cost will be incurred if long-running, low-resource requiring jobs are assigned to expensive VMs. We formulate the scheduling problem as follows. The goal of the scheduling problem, as defined in formula (1), is to minimize the cost while ensuring enough VMs for the incoming jobs. All the notations used in this

x_{ij_k}	whether job i is assigned to jth VM of type k, $x_{ik_j} \in \{0,1\}$
y_{j_k}	whether jth VM of type k is to-be-provisioned, $y_{k_j} \in \{0,1\}$
z_{ij}	whether job i is assigned to jth running VM, $z_{ij} \in \{0,1\}$
c_k	hourly cost of VM of type k
c_j	hourly cost of running VM j
w_j	capacity of running VM j
fc_k	full capacity of VM type k
r_i	resource consumption of job i
d_i	departure/end time of job i
s_j	The start time of VM j: the time when it started to boot
ld_j	latest departure time: the time that the final running job finish in VM j
ct	current time
M	number of newly arrived jobs, $\mathbb{M} = \{1, ..., M\}$
N	number of running VMs, $\mathbb{N} = \{1, ..., N\}$
K	number of types of VMs, $\mathbb{K} = \{1, ..., K\}$
$\lceil t \rceil$	math operation, divide time t by 3600 and get its ceil value.

Table 1. Overview of notations in Section 3.

section are listed in Table 1. An VM *to-be-provisioned* is identified by its identifier j and its type k, while a running VM is identified only by its identifier j.

Minimize

$$\sum_{k=1}^{K}\sum_{j=1}^{M}(y_{j_k} \times \lceil \underset{i\in\mathbb{M}}{max}(d_i \times x_{ij_k}) - ct \rceil \times c_k) + R \tag{1}$$

$$R = \sum_{j=1}^{N}(\lceil (max\{\underset{i\in\mathbb{M}}{max}(d_i \times z_{ij}), ld_j\} - s_j) \rceil \times c_j)$$

subject to

$$\sum_{i=1}^{M} z_{ij} \times r_i \leq w_j \qquad \forall j \in \mathbb{N} \tag{2}$$

$$\sum_{k=1}^{K}\sum_{i=1}^{M} x_{ij_k} \times r_i \leq fc_k \times y_{j_k} \qquad \forall j \in \mathbb{M} \tag{3}$$

$$\sum_{j=1}^{N} z_{ij} + \sum_{k=1}^{K}\sum_{j=1}^{M} x_{ij_k} = 1 \qquad \forall i \in \mathbb{M} \tag{4}$$

$$x_{ij_k} \leq y_{j_k} \qquad \forall i, j \in \mathbb{M}, \forall k \in \mathbb{K} \tag{5}$$

The cost of scheduling consists two parts: the cost of to-be-provisioned VMs and the cost of running VMs (defined by R). Each to-be-provisioned VM is charged between the current time (ct) and the latest departure time of its allocated jobs. The sum of the cost of running VMs, denoted by R, is defined similarly: each running VM is charged between the time it was started (s_j) and the latest departure time of its jobs (jobs that are running on VM and the jobs to-be-allocated to it).

This IPP is subject to a few constrains, which we describe in turn. Constraint (2) ensures that the allocated jobs in each VM cannot exceed the running VMs' capacity. Constraint (3) ensures that the allocated jobs in each *to-be-provisioned* VM can not exceed the VM's capacity. Constraint (4) ensures that each job is only

allocated to one VM. Constraint (5) ensures that each job will not be allocated to a VM that will not be provisioned. The decision variables x_{ij_k} and y_{kj} are binary. If the result of this IPP is that $y_{kj} = 0, \quad \forall k \in \mathbb{K}, \quad \forall j \in \mathbb{M}$, there will be enough VM capacity left to allocate all the future jobs. Otherwise, more VMs are needed. If $x_{ij_k} = 1$, job i will be allocated to the to-be-provisioned VM with identifier j type k.

3.3 Scheduling heuristics

We explore for *CoH* a large class of scheduling heuristic algorithms. They work as follows. While there are un-allocated jobs, each algorithm performs a loop consisting of four steps. Firstly, the algorithm sorts all the un-allocated jobs using *job selection* criteria and sorts all the VMs using *VM selection* criteria. Secondly, the algorithm picks the first un-allocated job. Thirdly, it picks the first VM which should have enough capacity left for the job. And then allocate the job to the selected VM. If such an VM does not exist, a new VM is provisioned according to *VM type selection* criteria and the job will be allocated in the next loop.

This general class of heuristic algorithms uses three criteria: *job selection*, *VM selection*, and *VM type selection* criteria. All *job selection* and *VM selection* criteria used in this work are listed in Tables 2 and 3. For *VM type selection*, we use a Cost-Efficient heuristic, which always chooses the VM with the largest *capacity/cost* value.

Most of the selection criteria we use in this work are simple, which allows them to be run online. We describe some of the criteria below. Latest arrival time (LA) sorts the VM according to the latest arrival time of jobs in each VM in decreasing order. Opposite to LT, Earliest arrival time (EA) sorts the VM by earliest arrival time of jobs in increasing order. Similar to LT, Latest departure time (LD) picks the VM which has the job that has the latest departure time; Earliest departure time (ED) does the opposite. The latest average arrival time (LAA) and earliest average arrival time (EAA) sorts VMs according to the average arrival time of their jobs in decreasing and increasing order, respectively. Close to full hour (CFH) makes use of the billing model of EC2; it always puts jobs on VM whose Billing Time Unit (BTU) is closest to be increased, while Far from Full Hour (FFH) is the opposite. In this work, the scheduling heuristic method specified by its job and VM selection criteria is uniquely identified as {*job selection*}-{*VM selection*}. For example, the FCFS-Rnd heuristic uses First-Come-First-Server (FCFS) for job selection, random (Rnd) for VM selection and the cost-efficient criteria for VM type selection.

4 Scheduling using reserved and on-demand instances

Cloud providers allow their users to reserve VM instances, long-term, for reduced cost. For instance, Amazon offers *reserved* instance, which can be rented for 1-3 years for a lower price than their on-demand counter-parts. When using reserved instead of on-demand instance, for the same VM configuration, users can pay a higher upfront cost (UF_i) for a lower hourly cost (C_i). Currently, there are three types of reserved instances supported in EC2: lightly utilized, medium-utilized, and

Name	Description
FCFS	First-come-first-server
RR	round-robin
LJF	Largest job first
SJF	Smallest job first
LTJF	Longest run-Time job first
STJF	Shortest run-Time job first

Table 2. Job selection criteria

Name	Description
Rnd	Random
LM	Largest capacity VM first
SM	Smallest capacity VM first
LA	Latest arrival time
EA	Earliest arrival time
LD	Latest departure time
ED	Earliest departure time
LAA	Latest average arrival time
EAA	Earliest average arrivial Time
CFH	Close to Full Hour
FFH	Far from full hour

Table 3. VM selection criteria

heavily utilized reserved instances. For the lightly utilized and medium-utilized instances, users need to pay an upfront cost and pay for each hour the VM is running. For the heavily utilized instances, users need to pay an upfront cost and pay for each hour during the reserved term even if the VM is not running. The hourly cost of Amazon EC2 instances are listed in Table 5. We present *CoH-R*, an extension of *CoH*, which uses reserved instances to reduce the operational cost. We describe the strategy of *CoH-R* in Section 4.1, then describe the method used to determine the number and types of reserved instances to be used in Section 4.2.

4.1 Policy Overview

Assuming that it is given an arbitrary amount of reserved instances, *CoH-R* makes use of these reserved instances as follows: the heavily utilized instances are always on, while the medium and lightly utilized instances are shut-down when they do not have any jobs running on them before their Billing Time Unit (BTU) is about to increase (m seconds before the BTU increases). Whenever *CoH* plans to start a new VM of type k, *CoH-R* firstly looks at medium-utilized instances of type k, and starts one of them if any is off. If no medium-utilized instance of type k exists, *CoH-R* tries to use a lightly utilized instance of type k. As a last resort, *CoH-R* uses on-demand instance of type k.

Having too few reserved instances will not benefit much from the reduced price; while reserving too much may actually increase operational cost. We do not seek to find the optimal number of reserved instances, because obtaining the optimal solution requires exact workload information of the entire reservation period (e.g., one year). Even if we can know the workload of the upcoming time period, obtaining the optimal solution via solving an IPP that takes the exact workload as input is computationally infeasible.

4.2 Determining the reservation plan

CoH-R only requires the workload distribution instead of exact information of the number of VMs needed at each time interval. For simplicity of analysis, we assume VM start-up and shut-down time are instantaneous (In experiment, we set the

start-up time as two minutes.). Assuming the number of VMs (resource demand) needed for the current time interval t is D_t, we can obtain the cost needed at each interval $B(D_t)$ as follows. If D_t is lower than the number of heavily-utilized instances (N_3), no other VMs will be needed, as the heavily utilized instances can provide enough computing resources. If D_t is high than N_3, but lower than the total number of heavily and medium-utilized instances ($D_t \leq N_3 + N_2$), *CoH-R* needs to provision $D_t - N_3$ medium-utilized instances. If D_t is higher than the sum of heavily and medium-utilized instances ($D_t > N_3 + N_2$) but lower than the total number of reserved instances, *CoH-R* needs to provision all the heavily and medium-utilized instances and $D_t - (N_3 + N_2)$ lightly utilized instances. Last, if D_t is higher than the sum of all reserved instances, *CoH-R* needs all the reserved instances and $D_t - (N_1 + N_2 + N_3)$ on-demand instances.

CoH-R obtains the number of reserved VMs needed, of each type, via finding the combination of number of reserved instances (N_i) that minimizes $\sum_{t=1}^{T} B(D_t) + \sum_{k \in \mathbb{K}} (UF_k \times N_k)$, where T is the number of intervals (e.g, hour) of a time period (e.g, year or month). Further, if the resource demand of each interval does not affect the other time intervals (in practice, most of the jobs' runtime is short, in the order of tens of minutes), the goal can be reformulated via only using the probability of VMs needed at each time interval as below, $(\sum_{i=0}^{M} Pr(D = i) \times B(i)) \times T + \sum_{k \in \mathbb{K}} (UF_k \times N_k)$, where $Pr(D = i)$ is the probability distribution of demand, and $\mathbb{K}$ is the set of reserved type, and M is the maximal number of VM needed.

We extend the above method which deal with one machine configuration only, to allow it to deal with multiple machine configurations. The goal is to find the number of reserved instances (N_{jk}) of machine configuration j and reserved type k, needed, to minimize the cost defined in formula (6).

$$(\sum_{i=0}^{M} Pr(D = i) \times B(i)) \times T + \sum_{j \in \mathbb{J}} \sum_{k \in \mathbb{K}} (UF_{jk} \times N_{jk}) \tag{6}$$

In formula (6), $\mathbb{J}$ is the set of machine configurations and UF_{jk} is the upfront cost of reserved instance of machine configuration j and reserved type k. The billing function $B(D)$ need to be changed to be the lowest cost to meet the demand D via finding the combination of reserved and on-demand instances to be used. $B(D) = \{Minimize \sum_{j \in \mathbb{J}} \sum_{k \in \mathbb{K}} (n_{jk} \times c_{jk}) + \sum_{j \in \mathbb{J}} (n_j^{od} \times c_j^o)\}$, where n_{jk} and c_{jk} are the number and the cost of the reserved instance with machine configuration j and reserved type k, respectively. n_j^{od} and c_j^o are the number and the cost of the on-demand instances of machine configuration j, respectively. The capacity offered by n_{jk} reserved instances and n_j^{od} on-demand instances should be enough to satisfy demand D.

5 Experimental results

In this section, we evaluate the performance of our proposed approaches using multiple real-world traces corresponding to two separate but popular domains: grid computing and online game hosting. Firstly, we compare *CoH* against various commonly used heuristics. Then, we evaluate *CoH-R*, which uses reserved instances to

further reduce cost, and compare it to *CoH*. Our results indicate that our proposed approaches can lead to significant lower cost than heuristics.

5.1 Experimental setup

We conduct experiments using three real-world workloads *LCG*, *Grid5000*, and *Dotalicious* which are taken from public workload archives [5, 8, 13]. *LCG* and *Grid5000* contain information about the computing activities of two grids while *DotaLicious* contains workload information of a game platform. We use the first year traces *Grid5000* and *Dotalicious*, and the full trace of LCG (13 days) as our input workloads. The common data we find in the above traces are, for each recorded job, its job id, the arrival time, and the departure time. The basic statistics of these workloads are listed in Table 4. Notably, the gaming server have similar runtime (CPU requirement) to Grid5000 jobs in the order of tens of minutes. Game servers are also computationally intensive, a result of having to perform virtual world physical simulation.

As not all the traces contain resource requirements for each job, we generate for each job resource requirements using 3 different methods: Heterogeneous, Constant-100, and Constant-10. For Heterogeneous workload, we generate each jobs's resource requirements as ten times a random number which is between 1 to 10. For Constant-100 method, each job's resource required is 100. For Constant-10 method, each job's resource required is 10. We only consider two instance types: `small` and `large`. We model a `small` EC2 instance's capacity as 100 and a `large` instance's capacity is 410. `Large` instance is more cost efficient than `small` instance. Their cost[3] are summarized in Table 5.

As running all the heuristics online is time consuming, we evaluate the heuristics by running simulation and pick the heuristics that have good performance as alternative method to compete with the solution obtained by solving IPP. We find that none of heuristics can perform always best, across all scenarios, and find that the job selection criteria does not have a significant impact on cost but VM selection criteria does have an important impact on cost. We pick FCFS-SM when the input workload is heterogeneous, and use FCFS-LD and FCFS-CFH when the workload is homogeneous (Constant-10 and Constant-100).

All the experiments are conducted using our own simulator[4] and repeated at least 10 times. We set the acquisition time of an VM to two minutes and the scheduler is executed every 10 seconds. We use IBM CPLEX to solve the formulated IPP when the number of jobs to be scheduled is lower than 50 and set the time limited as two seconds. As our methods have proactively provision VMs for all the jobs, the wait time of each job is zero. We evaluate one metric, the rental cost. The rental cost is the price paid to cloud providers for all the rented computing resource. We focus on cost because it is a major barrier for cloud adoption.

For calculation of the utility of all the methods, we compare the lower-bound for cost against actually paid cost. The lower bound for cost is calculated by assuming

[3] `http://aws.amazon.com/ec2/pricing/`
[4] `http://www.pds.ewi.tudelft.nl/~siqi/simulator.htm`

that we have an ideal computer that it can vertically scale to the any of the desired capacity. The vertical scaling takes zero time and the VM is charged by its actual usage of resource which scales linearly with its capacity. So the optimal cost can be computed as $[\sum_{i=1}^{N} r_i \times (d_i - a_i)] \div w_k \times c_k, \quad \forall i \in \mathbb{N}$, where N is the total number of jobs, and w_k and c_k are the capacity and the cost of the most cost-efficient VM, respectively.

Trace	#jobs	average runtime [s]	duration	source
Grid5000	200,450	2728	May 2004 - May 2004	Grid workload archive [13]
LCG	188,041	8971	Nov 2005 - Dec 2005	Parallel workload archive [5]
DotaLicious	109,251	2231	Apr 2010 - Apr 2011	Game trace archive [8, 9]

Table 4. Overview of traces.

	Small (hourly, upfront) [$]	Large (hourly, upfront) [$]
On demand	(0.065, 0)	(0.26, 0)
Lightly utilized	(0.039, 69)	(0.156, 276)
Medium utilized	(0.024, 160)	(0.096, 640)
Heavily utilized	(0.016, 195)	(0.064, 780)

Table 5. Overview of cost of EC2 instances: Small and Large.

5.2 Results

We first evaluate *CoH* against various heuristic methods. Figure 1 shows the average experiment results using *Grid5000* and *LCG* datasets, respectively. The error bars are the standard deviation. Figure 1 shows the lower bound for cost (LB), and results for FCFS-SM, FCFS-CFH, FCFS-LD, and *CoH*, from left to right; grouped by type of workloads. We find that *CoH* performs better than any of the heuristics. For the *Grid5000* dataset, *CoH* can obtain about 20% to 40% lower cost than any heuristic. For the *LCG* dataset, *CoH* can obtain 5% to 20% lower cost. This indicates that CoH can find better combinations of VMs, and better mapping between jobs and VMs.

The cost obtained through *CoH* is about 1.1 to 1.6 times higher than LB. The utilization of *CoH*, that is, the average use of leased VMs, ranged from 90% to 63%. We identify three reasons why *CoH* is higher than the lower bound (LB): Firstly, our scheduler is run online, thus not having all the necessary information. Secondly, the billing model of the cloud: a fractional consumption of a VM's capacity is charged as the fully busy VM. Thirdly, the boot-up time of VM is not negligible. One possible way to lower the gap between LB and *CoH* is to allow the jobs to wait for better scheduling opportunity, so that scheduler can pack more jobs in the same VM instead of starting a VM for each short job. This approach would be particularly effective during bursts in the workload.

We evaluate *CoH-R* using the *Dotalicious* and *Grid5000* datasets. The results are shown in Figure 2. We do not evaluate *LCG* dataset because it lasts for only 13 days (less than the minimal reservation period of EC2). We compare in Figure 2: FCFS-CFH, *CoH*, CoH-oneType, and *CoH-R*. CoH-oneType is a variation of *CoH-R* which only uses heavily-utilized instance. For the *Dotalicious* dataset, *CoH-R* and CoH-oneType obtain lower cost than *CoH*. *CoH-R* can obtain lower cost than CoH-oneType, because it takes advantage of the cost reduction and flexibility provided by different reserved types. The result obtained by *CoH-R* using the *Dotalicious* dataset is about 13% to 20% lower than *CoH* and about 30% to 60% lower than

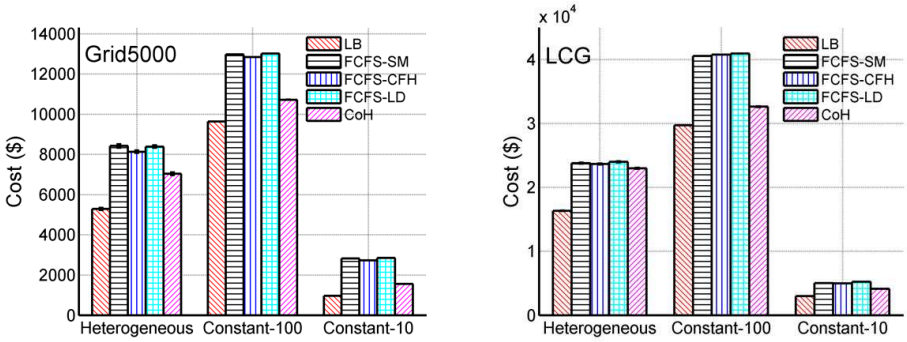

Fig. 1. Cost of various scheduling methods: Grid5000 (Left) and LCG (Right).

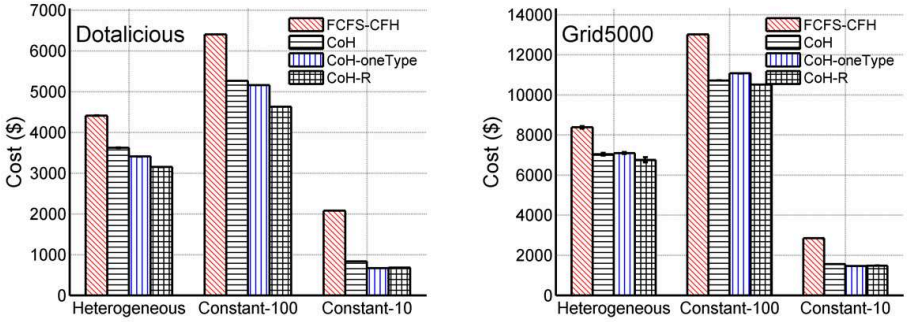

Fig. 2. Effect of using reserved instances: Dotalicious (Left) and Grid5000 (Right).

FCFS-CFH. For the *Grid5000* dataset, the performance of *CoH-R* obtain about 3% to 5% lower cost than *CoH*, but still about 20% to 50% lower cost than the heuristic. The reason why *CoH-R* only obtains a small improvement on *Grid5000* is because *Grid5000* contains busty workloads with short jobs, and some very long jobs. As *CoH-R* always schedules jobs to VMs as soon as the jobs arrive in the system, this cause some long jobs to run on on-demand instances instead of the cheaper reserved instances. In summary, *CoH-R* can obtain about 20% and up to 60% lower cost than the heuristic. *CoH-R can obtain significantly lower cost than heuristics which use on-demand instances only.*

6 Related work

A significant body of work has already focused on cloud resource scheduling from a cloud provider's perspective [10,22,27,31]. In this context, the common goals are to reduce the storage/electricity cost and to improve platform utilization. In contrast, in this study we schedule resources from a broker's perspective, with the goal to minimize the rental cost.

Previous studies have focused on provisioning and allocation of cloud resources, under various constraints. In contrast to these studies, which we describe in the following, we consider multiple instance types, billing models and heterogeneous workload. Closest to our work, Genaud and Gossa [7] evaluate provisioning heuristics for on-demand resources. Villegas et al. [28] conduct a performance-cost analysis

of scheduling policies for IaaS Cloud. Deng [4] et al develop a portfolio scheduler. Oprescu and Kielmann [20] schedule bag-of-tasks on clouds focusing on budgets and runtime. They formulate the provisioning problem as a Bounded Knapsack Problem and allocate jobs to VMs round-robin. Mao et al. [15] propose a linear program for provisioning, and allocate jobs randomly to VMs. Sharma et al. [24] use on-demand instances and use migration but only for homogeneous workloads.

Hong et al [11] use a method to determine number of reserved instances of one reservation type. We show in our experiments that it is necessary to use multiple reserved instance types to reduce cost. Chaisiri [3] propose an algorithm to determine the number and types of reserved VM to be used by solving a stochastic IPP to minimize expected cost. They limit the on-demand instances can be only provisioned in specific provision phase, while we proactively provision VM at any necessary time. Ostermann and Prodan [21], and Song et al. [25] use spot-instance to reduce cost. Their work complement ours.

7 Conclusion

It is challenging to select among machine configurations and billing options offered by clouds to fit user demand while reducing operational cost. In this work, we propose *CoH*, a Cloud-base, online, Hybrid scheduling policy which make uses of multiple machine configurations to plan enough capacity for users with less cost. We formulate the resource provisioning and the job allocation problems as Integer Programming Problems (IPP). To obtain the scheduling decision online, *CoH* limits the time of exploration for a solution and only obtains an intermediate IPP solution. *CoH* makes scheduling decision by picking the best among the solution of IPP and various heuristics; thus, *CoH* operates as a portfolio scheduler. Further, we propose *CoH-R*, a policy that makes use of both on-demand and reserved instances to reduce cost. Via simulation using real-world traces, we show that our approaches can lead to significant lower cost than heuristics while operating online. We plan to investigate the wait-time and rental cost trade-off for bursty workload comprised of many short jobs.

Acknowledgement The work is supported by CSC-TUD grant, the National Basic Research Program of China (973) under grant No.2011CB302603, NSFC under grant No.60903042, and by the STW/NOW Veni grant 11881.

References

1. de Assuncao, M.D., di Costanzo, A., Buyya, R.: Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters. pp. 141–150. HPDC '09
2. Ben-Yehuda, O., Schuster, A., Sharov, A., Silberstein, M., Iosup, A.: Expert: Pareto-efficient task replication on grids and a cloud. In: IPDPS 2012. pp. 167–178
3. Chaisiri, S., Lee, B.S., Niyato, D.: Optimization of resource provisioning cost in cloud computing. Transactions on Services Computing 2012 pp. 164–177
4. Deng, K., Verboon, R., Iosup, A.: A Periodic Portfolio Scheduler for Scientific Computing in the Data Center. In: JSSPP (2013)
5. Feitelson, D.: Parallel Workloads Archive. `http://www.cs.huji.ac.il/labs/parallel/workload/`

6. Folling, A., Hofmann, M.: Improving scheduling performance using a q-learning-based leasing policy for clouds. In: Euro-Par 2012, pp. 337–349
7. Genaud, S., Gossa, J.: Cost-wait trade-offs in client-side resource provisioning with elastic clouds. In: CLOUD 2011
8. Guo, Y., Iosup, A.: The game trace archive. In: NETGAMES 2012
9. Guo, Y., Shen, S., Visser, O., Iosup, A.: An Analysis of Online Match-Based Games. MMVE2012
10. Hadji, M., Zeghlache, D.: Minimum cost maximum flow algorithm for dynamic resource allocation in clouds. In: CLOUD 2012. pp. 876–882
11. Hong, Y.J., Xue, J., Thottethodi, M.: Selective commitment and selective margin: Techniques to minimize cost in an iaas cloud. In: ISPASS 2012. pp. 99–109
12. Huberman, B.A.: An Economics Approach to Hard Computational Problems. Science 275, 51–54 (1997)
13. Iosup, A., Li, H., Jan, M., Anoep, S., Dumitrescu, C., Wolters, L., Epema, D.H.J.: The grid workloads archive. FGCS 2008 24(7), 672–686 (Jul.)
14. Iosup, A., Ostermann, S., Yigitbasi, N., Prodan, R., Fahringer, T., Epema, D.: Performance analysis of cloud computing services for many-tasks scientific computing. TPDS (2010)
15. Mao, M., Li, J., Humphrey, M.: Cloud auto-scaling with deadline and budget constraints. In: GRID 2010. pp. 41–48
16. Marshall, P., Keahey, K., Freeman, T.: Elastic site: Using clouds to elastically extend site resources. In: CCGrid 2010. pp. 43–52
17. Murphy, M., Kagey, B., Fenn, M., Goasguen, S.: Dynamic provisioning of virtual organization clusters. In: CCGrid 2009. pp. 364–371
18. Nae, V., Iosup, A., Prodan, R.: Dynamic resource provisioning in massively multiplayer online games. TPDS 2011 22(3)
19. Nicolae, B., Cappello, F., Antoniu, G.: Optimizing multi-deployment on clouds by means of self-adaptive prefetching. In: Euro-Par 2011. pp. 503–513
20. Oprescu, A., Kielmann, T.: Bag-of-tasks scheduling under budget constraints. In: CloudCom 2010. pp. 351–359
21. Ostermann, S., Prodan, R.: Impact of variable priced cloud resources on scientific workflow scheduling. pp. 350–362. Euro-Par'12
22. Ren, S., He, Y., Xu, F.: Provably-efficient job scheduling for energy and fairness in geographically distributed data centers. In: ICDCS 2012. pp. 22–31
23. Schwiegelshohn, U,. Badia, Rosa M., Bubak , M. et al: Perspectives on grid computing. FGCS 2010 26(8)
24. Sharma, U., Shenoy, P., Sahu, S., Shaikh, A.: A cost-aware elasticity provisioning system for the cloud. In: ICDCS 2011. pp. 559–570
25. Song, Y., Zafer, M., Lee, K.W.: Optimal bidding in spot instance market. In: INFOCOM 2012. pp. 190 –198
26. Stillwell, M., Vivien, F., Casanova, H.: Dynamic fractional resource scheduling for hpc workloads. In: IPDPS 2010
27. Tordsson, J., Montero, R.S., Moreno-Vozmediano, R., Llorente, I.M.: Cloud brokering mechanisms for optimized placement of virtual machines across multiple providers. Future Gener. Comput. Syst. 28(2) (2012)
28. Villegas, D., Antoniou, A., Sadjadi, S.M., Iosup, A.: An analysis of provisioning and allocation policies for infrastructure-as-a-service clouds. In: CCGrid 2012
29. Warneke, D., Kao, O.: Exploiting dynamic resource allocation for efficient parallel data processing in the cloud. TPDS 2011 pp. 985 –997
30. Webb, J.: How the cloud helps Netflix. [Online] Available: `http://radar.oreilly.com/2011/05/netflix-cloud.html` (May 2011)
31. Zhang, T., Du, Z., Chen, Y., Ji, X., Wang, X.: Typical virtual appliances: An optimized mechanism for virtual appliances provisioning and management. Journal of Systems and Software 84(3), 377 – 387 (2011)