



Delft University of Technology
Parallel and Distributed Systems Report Series

**An Analysis of Provisioning and Allocation Policies for
Infrastructure-as-a-Service Clouds: Extended Results**

David Villegas, Athanasios Antoniou,
Seyed Masoud Sadjadi, and Alexandru Iosup
dvi11013@cs.fiu.edu, A.Iosup@tudelft.nl

Draft, Dec 2011.

report number PDS-2011-009



ISSN 1387-2109

Published and produced by:
Parallel and Distributed Systems Group
Department of Software and Computer Technology
Faculty Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

Information about Parallel and Distributed Systems Report Series:
reports@pds.ewi.tudelft.nl

Information about Parallel and Distributed Systems Section:
<http://pds.ewi.tudelft.nl/>

© 2011 Parallel and Distributed Systems Group, Department of Software and Computer Technology, Faculty Electrical Engineering, Mathematics, and Computer Science, Delft University of Technology. All rights reserved. No part of this series may be reproduced in any form or by any means without prior written permission of the publisher.





Abstract

Today, many commercial and private cloud computing providers offer resources for leasing under the infrastructure as a service (IaaS) paradigm. Although an abundance of mechanisms already facilitate the lease and use of single infrastructure resources, to complete multi-job workloads IaaS users still need to select adequate provisioning and allocation policies to instantiate resources and map computational jobs to them. While such policies have been studied in the past, no experimental investigation in the context of clouds currently exists that considers them jointly. In this paper we present a comprehensive and empirical performance-cost analysis of provisioning and allocation policies in IaaS clouds. We first introduce a taxonomy of both types of policies, based on the type of information used in the decision process, and map to this taxonomy eight provisioning and four allocation policies. Then, we analyze the performance and cost of these policies through experimentation in three clouds, including Amazon EC2. We show that policies that dynamically provision and/or allocate resources can achieve better performance and cost, but that the lack of adaptation to the specific pricing model and technical capabilities of the used cloud can significantly reduce cost-effectiveness. We also look at the interplay between provisioning and allocation, for which we show preliminary results.



Contents

1	Introduction	4
2	System Model	5
2.1	Workload Model	5
2.2	Resource Model	5
3	Provisioning and Allocation Policies	6
3.1	Provisioning Policies	6
3.2	Allocation Policies	7
4	Experimental Setup	9
4.1	The SkyMark Empirical Performance Evaluation Tool	9
4.2	Experimental Environments	9
4.3	Workloads	10
4.4	Performance, Cost, and Compound Metrics	10
5	Experimental Results	11
5.1	Provisioning Policies	11
5.2	Allocation Policies for Static Resources	12
5.3	Effects of job size distribution in on-demand policies	13
5.4	Policy interactions	14
6	Related Work	16
7	Conclusion and Future Work	19

List of Figures

1	The cloud ecosystem.	5
2	Workload Makespan (WMS).	12
3	Job Slowdown (JSD).	12
4	Workload speedup (SU).	12
5	Workload Slowdown Inf. (SU_{∞}).	12
6	Actual cost (C_a).	13
7	Charged cost (C_c).	13
8	Cost efficiency (C_{eff}).	13
9	Utility (U).	13
10	Instances over time for the provisioning policies with the <i>Uniform</i> workload on DAS4.	15
11	Instances over time for the provisioning policies with the <i>Increasing</i> workload on DAS4.	16
12	Instances over time for the provisioning policies with the <i>Bursty</i> workload on DAS4.	17
13	Average Job Slowdown for Allocation policies.	18
14	Interarrival time for periodic distribution.	18
15	Slowdown and cost for job runtime ratios.	18
16	Uniform, Periodic, and Bursty interarrival times.	19
17	Slowdown and cost for groups of policies.	20
18	VM provisioning for Uniform, Periodic, and Bursty workloads.	21

List of Tables

1	Overview of provisioning policies.	6
2	Overview of allocation policies.	7
3	Overview of the experimental environments.	9
4	Specification of VM Instances across environments.	10
5	Charged cost, cost efficiency and utility for policies on DAS4.	11
6	Charged cost, cost efficiency and utility for policies on FIU.	14
7	Charged cost, cost efficiency and utility for policies on EC2.	14

1 Introduction

Recent advances [1, 2] in the high-speed yet low-cost interconnection of off-the-shelf computational and storage resources have facilitated the creation of data centers of unprecedented scale. As a consequence, a fundamental shift is beginning to occur in the way computational resources are provisioned and allocated by our society, from traditional ownership to Infrastructure-as-a-Service (IaaS) clouds—leasing and releasing virtualized resources. Although hundreds of commercial IaaS providers exist, to transition to IaaS clouds users still need detailed understanding of achieved performance and incurred cost. In particular, potential IaaS users need to understand the performance and cost of resource provisioning and allocation policies, and the interplay between them. To address this need, we conduct an empirical analysis of resource provisioning and allocation policies for IaaS clouds.

The basic operation of an IaaS cloud is to temporarily instantiate on-demand virtual machines (VMs)—of pre-agreed computing power and memory size, operating system and provided libraries, and, usually, some measure of Quality of Service (QoS). Providers host the resources shared by all users and can take advantage of economies of scale by leasing their physical infrastructure as virtualized resources to many different classes of users. Users provision, that is, acquire and release, resources based on their actual needs, only when, where, and for how long needed; they can allocate the provisioned resources according to the specific requirements of the workloads at hand. This model makes IaaS clouds a flexible solution that reduces or completely eliminates the need for acquiring and managing costly physical resources, but also introduces the need to consider *online* the trade-off between performance (more resources) and cost.

The provisioning and allocation policies can have an important impact on the traditional performance metrics, from workload makespan to individual job slowdown [3]. Since instantiating a large number of VMs is simple, over-provisioning can incur a substantial yet unnecessary cost; when the allocation policy is inefficient, a dynamic provisioning policy may lease resources that remain largely unused [4]. The pricing scheme of the IaaS cloud, which may include hourly charging periods and discounts for first-time use, may lead to different cost gains than expected from actual resource consumption.

The performance-cost trade-off has been extensively studied in the context of grids, mainly from the perspective of users also acting as providers and looking for economic gain [5] or for sharing fairness [6]. Moreover, more traditional resource managers such as Condor support [7, 8], through versatile job specification languages, complex criteria for the selection of resources. Several studies [3, 9, 4] have approached this trade-off in simulation, in the context of clouds. However, until now no study has investigated in practice the impact of the provisioning and allocation policies that users can employ, and of the interplay between these policies, in the context of clouds. The need for empirical evaluation stems from recent results [10, 11, 12] in cloud performance evaluation, which show that cloud performance is much lower and more variable than considered in simulation models. In this work we propose a comprehensive investigation of provisioning and allocation policies for IaaS clouds. Our main contribution is threefold:

1. We identify eight provisioning and four allocation policies that can realistically be applied for managing workloads in IaaS clouds (Section 3);
2. We conduct an empirical investigation using three IaaS clouds, including the services provided by the commercial IaaS Amazon EC2 (Section 4);
3. We analyze empirically and, only for the otherwise expensive experiments, in simulation the performance and cost of resource provisioning and allocation policies, and the interplay between these two types of policies (Section 5).

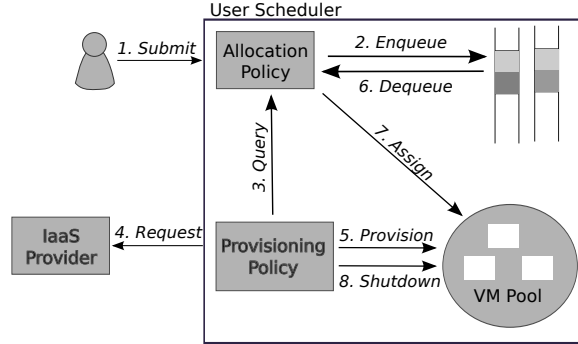


Figure 1: The cloud ecosystem.

2 System Model

In this section we present the system model used throughout this work.

2.1 Workload Model

Although desirable, it is not yet possible to define a realistic workload for IaaS clouds, due to the scarcity of public workload traces or common practice reports. The Parallel and the Grid Workload Archives [13] provide in total tens of workload traces, but it is yet unknown if the users of these environments will migrate to IaaS clouds [11]. We have already shown in our characterization of over fifteen grid workloads [14] two trends: the disappearance of tightly-coupled parallel jobs in favor of Bags of Tasks (BoTs), and the decrease of the amount of work (runtime) of each task.

For this work, we consider synthetic, BoT-based workloads with runtimes typical for data mining and semi-interactive processing, that is, several minutes [15, 16]. In our workload model, jobs are CPU-bound and their runtime is dependent on the speed of the (virtual) processor where they are executed.

2.2 Resource Model

In our system model, resources are provisioned exclusively from IaaS clouds. Although hybrid local-cloud systems still exist, we anticipate with this work the moment when buying and maintaining local resources will be anachronistic. The cloud ecosystem investigated in this work, which is comprised of the IaaS cloud provider(s) and user(s), is depicted in Figure 1. We assume that users send their workloads to a scheduler, which enqueues jobs and assigns them to the pool of available resources based on an *allocation policy*. A system component manages the pool of resources via a *provisioning policy*, that is, a policy that decides when to lease and to release resources from IaaS clouds. This component can query the state of the allocation, such as the queue size or the average waiting time for jobs.

We model the operation of IaaS clouds based on Amazon EC2, a popular commercial IaaS. We assume that a provisioning request issued to the IaaS cloud will incur delays that depend on the process used by the IaaS to select, lease-and-boot, and release (shut down) a VM instance. Last, we assume that VMs have a cost associated to their operation, with a cost model that is proportional, possibly non-linearly, to the runtime of the VM instance, and not counting the time required to boot or shut down the instance.

Policy	Dynamic	Trigger	Job Duration	Increase Param.
Startup	No	—	—	—
OD-S	Yes	Queue size	No	1
OD-G	Yes	Queue size	No	n
OD-ExecTime	Yes	Exec. time	Yes	1
OD-ExecAvg	Yes	Exec. time	No	1
OD-ExecKN	Yes	Exec. time	No	1
OD-Wait	Yes	Wait time	No	1
OD-2Q	Yes	Queue size	Partial	1

Table 1: Overview of provisioning policies.

3 Provisioning and Allocation Policies

We present in this section the provisioning and allocation policies considered for this work. Although we envision that future policies will adapt to changing workload, evolving resources, and complex Service Level Agreements, we focus in this work on the simple policies that may be realistically employed in practice, today.

3.1 Provisioning Policies

We consider for this work eight provisioning policies; we summarize their properties in Table 1 and describe them next. Overall, we identify two main classes of provisioning policies, static and dynamic. For the class of dynamic provisioning policies, we investigate three criteria for deciding when the policies are called (*triggers*): the current size of the queue, the accumulated execution time of the queued jobs, and the total waiting time among queued jobs. Some of the provisioning policies considered here require information about the duration of jobs, which is not always provided by the user; we consider, for these policies, alternatives that estimate this information from historical records. The last policy in Table 1 requires partial information, that is, it only requires classification information—a job can be either small or long. We now describe the provisioning policies, in turn:

1. Startup

is a provisioning policy that leases all the needed VM instances during the system startup. While this ensures that no delay will occur due to waiting for new resources to be leased, this policy is not flexible—whether jobs arrive or not in the system, VM instances are still leased and paid; also, this policy cannot cope well with bursty system overloads.

2. On-Demand, Single VM (OD-S)

is a naïve dynamic provisioning policy that leases a new VM instance for each job that cannot be assigned to a resource. VMs are shut down when they are not used for a certain duration, determined as a parameter; by setting this parameter to 0 seconds, VMs are released immediately after the job is completed. In general, this policy can lead to *thrashing*, that is, frequent leasing and releasing of VM instances.

3. OD-Geometric (OD-G)

is a dynamic policy that extends OD-S with geometric provisioning, that is, when new instances are needed this policy provisions n^0, n^1, n^2, \dots machines in successive leases, where n is the *increase parameter*. Similarly,

Policy	Uses Queue	Job Duration	Provisioning-Aware
FCFS	Yes	No	No
FCFS-NW	No	No	No
SJF	Yes	Yes	No
FCFS-2Q	Yes (2)	Partial	Yes

Table 2: Overview of allocation policies.

this policy releases increasing amounts of instances. The decision of (re)leasing instances is taken periodically, when the number of available VMs falls below a threshold, or the number of tasks in the system queue exceed a limit; here, we use a period of 20 seconds and (re)lease again whenever there is at least one queued job/idle VM instance.

4. OD-ExecTime

is a dynamic provisioning policy that uses the future execution time of queued jobs, which is assumed to be known *a priori*, to decide on leasing or releasing VM instances. The decision to lease is adaptive to the cloud, in that the execution time of queued jobs must exceed the average time needed to provision and boot a VM instance (as observed for previously leased VMs) by a specified factor, which is set in this work to 5.

5. OD-ExecAvg

is similar to OD-ExecTime, but the execution time for each queued job is *estimated* as the average execution time of all the jobs that have already completed. An initial prediction of the average job run-time must be provided by the user.

6. OD-ExecKN

is similar to OD-ExecAvg, but uses a predictor based on [17]. For each job in the queue, OD-ExecKN acquires its k -nearest neighbors based on the job input parameter size. Then, the estimated execution time for a job is the average over this set of k neighbors.

7. OD-Wait

is similar to OD-ExecTime, but it considers the waiting times of queued jobs instead of their future execution time.

8. OD-2Q

is a provisioning policy that works in conjunction with the FCFS-2Q allocation policy (Section 3.2). To minimize the trashing of VMs for short running jobs, we define a bi-queue on-demand provisioning policy that leases VM instances and assigns them to one of two pools. The two pools effectively implement two OD-S queues with separate idle time parameters; here, we use longer idle times for VMs that will run short jobs, so that thrashing is reduced.

3.2 Allocation Policies

We consider for this work four allocation policies; their properties are summarized in Table 2. Most policies we investigate use one queue, but we also consider policies that use two or no queue. Similarly to our approach

for provisioning policies, we consider here allocation policies that may require (partial) information about the job duration. The last policy in Table 2 is *provisioning-aware*, that is, it works in conjunction with the OD-2Q provisioning policy. We now describe the allocation policies, in turn:

1. First-Come, First-Served (FCFS)

is a traditional allocation policy that assigns one task per resource in the order in which the tasks have been submitted to the system.

2. FCFS-NoWait (FCFS-NW)

is an extension to FCFS where jobs are not queued when no available VMs exist. Instead, this policy assigns jobs to VMs that are already running other jobs, round robin. This policy eliminates the wait time, but may introduce bottlenecks in the execution of jobs.

3. Shortest-Job First (SJF)

is a traditional allocation policy that gives priority to shorter jobs, assuming there is some information about or estimation of their actual duration. Although it alleviates the FCFS problem of short jobs waiting for the allocation of longer jobs with earlier arrival time, it can lead to starvation for long jobs.

4. FCFS-MultiQueue

is an extension to FCFS where several FCFS queues, one for each range of job durations, are maintained. The simplest case considered here, FCFS-2Q, has two queues, one for short jobs and another for long jobs. Although an estimation of the runtime is necessary for this policy, it is enough to have partial knowledge of it to classify jobs. This policy can work in conjunction with a provisioning policy that divides VMs into pools for short and long running jobs, such as OD-2Q (introduced in Section 3.1).

System	Hardware Spec	VIM/Hypervisor	Max VMs
DAS4 Delft	8 dual quad-core 24 GB RAM	OpenNebula 3.0 / KVM (Full, HVM)	64
FIU	8 Pentium 4 5 GB Memory	OpenNebula 2.2 / XEN (Para., No HVM)	7
Amazon EC2 eu-west-1	-	- / XEN (Para., No HVM)	20

Table 3: Overview of the experimental environments.

4 Experimental Setup

In this section we discuss our experimental setup, in turn, the SkyMark empirical performance evaluation tool, the used testbeds, the workloads, and the employed metrics.

4.1 The SkyMark Empirical Performance Evaluation Tool

Unless otherwise specified, we have conducted the experiments presented in this work in *real* environments. To this end, we have implemented SkyMark, a performance evaluation framework that enables the generation, submission, and monitoring of complex workloads to IaaS cloud environments. SkyMark extends GrenchMark [18] and C-meter [19], our prior IaaS performance evaluation framework, with provisioning and allocation policies, new workloads, and new analytical capabilities. SkyMark currently supports IaaS clouds that implement Amazon EC2’s interface, including all deployments using Eucalyptus, but interfaces to other clouds can be easily plugged-in to the tool. SkyMark also supports the XML-RPC system interface of OpenNebula.

The experimental process uses SkyMark as follows: The user provides a workload description file which is used to generate a real or synthetic workload. The workload is then submitted to a cloud, using pre-specified provisioning and allocation policies. Performance statistics are gathered throughout the execution of the workload and then stored in a database. Last, the database is used for post-experiment analysis. SkyMark effectively plays the role of both the user and the policies depicted in Figure 1.

Since using a real testbed is constrained by actual resource and budget availability, we have also developed a discrete event simulator that duplicates SkyMark functionality. The simulator reads a workload description and generates events for job arrivals, VM lifecycle, and pluggable allocation and provisioning policies.

4.2 Experimental Environments

We have performed experiments on three *real* IaaS clouds with different resources, middleware, and virtualization systems. The three used systems are: the Delft cluster of DAS-4 ¹, a six-cluster wide-area distributed system in the Netherlands; a cluster at Florida International University (FIU); and the Amazon EC2 commercial IaaS. The properties of the three IaaS clouds used in this work are summarized in Table 3. The used VM instance characteristics are described in Table 4.

¹<http://www.cs.vu.nl/das4/>

System	OS	Instance Specification			
		Virt. cores	Mem.	Disk	Platform
DAS4	CentOS 5.4 x86-64	1	1GB	5GB	64-bit
FIU	CentOS 5.3 i386	1	512MB	3GB	32-bit
EC2	RHEL 6.1 i386 ID:ami-9289bae6	1	1.7GB	160GB	32-bit

Table 4: Specification of VM Instances across environments.

4.3 Workloads

We have used the following workloads, each hour-long:

1. **Uniform:** A steady stream of tasks throughout the experiment; uses a Poisson arrival distribution. The average system load is around 70%.
2. **Increasing:** The workload intensity increases over time, in steps. The average system load is around 50%.
3. **Bursty:** The workload features short spikes of intense activity amid long periods of mild or moderate activity. The average system load is around 15%; the maximum load is, for a few minutes, 170%.

For the simulated results in section 5, we use the Uniform and Bursty workloads, and add a new one, Periodic, following a periodic increasing and decreasing arrival pattern. For these workloads we also change the job durations. We set the simulated VM boot time to 10 minutes, which is an average value we have observed in private clouds (DAS-4 and FIU). More details are discussed in the corresponding section.

The jobs that comprise the workloads are synthetic and have an average execution time of 47 seconds with a standard deviation of 41.1, as measured on DAS-4 when jobs were run independently. The reason for selecting short job durations were discussed in Section 2.1.

4.4 Performance, Cost, and Compound Metrics

We use a variety of performance, cost, and compound metrics to analyze the impact of provisioning and allocation policies. For performance metrics, we look at the traditional [20] metric of *workload makespan* (WMS) and at the *average job slowdown* (JSD), defined per job as the ratio of the actual runtime in the cloud and the runtime in a dedicated environment; the former metric is useful for BoT users, while the latter is useful for users of semi-interactive or individually-meaningful jobs. We also look at the workload speedup, measured against a single node (SU) and against an infinitely-large system (SU_{∞}); SU has values above 1 and is, theoretically, not bound, whereas SU_{∞} has values below 1.

We use two cost metrics. The *actual cost* (C_a) is defined as the sum of consumed resources, here, CPUtime. The *charged cost* (C_c) is the price charged by the provider, here, using the Amazon EC2 pricing model of charging CPUtime consumption in increments of one hour.

We define two compound metrics, cost efficiency and utility. We define *cost efficiency* (C_{eff}) as the ratio of the charged and actual cost; lower values are better, with a value of 1 indicating fair pricing, and values below 1 indicating dumping prices or economies-of-scale. We define *utility* (U) as a compound metric that rewards low performance overheads and low cost: $U = \frac{SU}{C_c}$. The values for utility are normalized on the Startup provisioning policy.

5 Experimental Results

In this section we perform a set of experiments to explore the effects of different policies and their interactions on different systems and under varying conditions. We want to determine how policies perform, when it is better to use ones versus others, and which allocation and provisioning policies work better when used together.

5.1 Provisioning Policies

We first explore the effect of the provisioning policies. To this end, we use the same allocation policy, FCFS, coupled in turn with each one of the provisioning policies. We show the results in Figures 2-12 and in Tables 5-7.

Figures 2-5 present the workload makespan (WMS), the job slowdown (JSD), the workload speedup against a single node(SU) and the workload slowdown against an infinitely large system (SU_{∞}) respectively. From these figures, it is apparent that **Startup** always achieves the best performance. **OD-S** has similar performance for the uniform workload, but is not as good for the variable workloads. From the threshold-based policies, **QueueWait** usually performs better than the rest, because it reacts faster to load variation. **ExecTime** and its variants have similar performance, with **ExecTime** usually performing better, since **ExecAvg** and **ExecKN** do not have exact job runtime information.

The actual cost (C_a) and charged cost (C_c) are presented in Figures 6 and 7 respectively. Even though **Startup** incurs the highest actual cost, since it acquires the full set of resources from the start until the end of the experiments, it is **OD-S** that costs the most according to Amazon's billing scheme. With **OD-S** the VMs are started and stopped reactively to individual job arrivals. The group of threshold-based policies and especially the **Exec** family of policies significantly reduce the cost of workload execution. The cost reduction becomes bigger for the increasing and bursty workloads. **QueueWait** appears to have similar actual cost to the **Exec** policies, however the charged cost is higher, especially for the variable-load workloads.

Figures 8 and 9 show the cost efficiency (C_{eff}) and utility (U) for the set of provisioning policies. The dynamic policies hold on to resources for shorter periods of time than **Startup**, especially for bursty workloads. This leads to a worse cost efficiency value. However, they do achieve better utility scores, which means that they provide a better performance-cost trade-off. The charged cost(C_c), cost efficiency (C_{eff}) and utility (U) values for the three clouds are presented in Tables 5-7.

More insight about the provisioning policies can be gained from Figures 5-7. Here, the number of requested and acquired resources is plotted over time, for the DAS4 cloud, when under the three different types of workloads. **OD-S** requests and releases resources very often, leading to VM thrashing. Particularly visible under the increasing and bursty workloads is that **OD-S** often releases resources before they even become accessible. **QueueWait** reacts faster to load variation, thus leading to higher cost. The **Exec** group of policies, and especially the two run-time estimating policies, have slower reaction to load variation. The discrepancy between **ExecTime** and its variants is caused by a bad initial prediction of the job run-time, that is manually configured prior to the start of the experiments. For the bursty workload, the inaccurate prediction of the run-time leads to better behavior for the two estimation-based policies.

	ChargedCost			CostEfficiency			Utility		
	Uniform	Increasing	Bursty	Uniform	Increasing	Bursty	Uniform	Increasing	Bursty
Startup	128	128	127	1.7	1.7	1.8	1.0	1.0	1.0
ExecTime	77 (-40%)	52 (-59%)	52 (-59%)	1.6 (-3%)	1.8 (+5%)	2.8 (+61%)	1.4 (+37%)	2.0 (+105%)	2.3 (+127%)
ExecAvg	81 (-37%)	43 (-66%)	22 (-83%)	1.7 (+2%)	1.5 (-12%)	1.3 (-27%)	1.2 (+19%)	2.1 (+114%)	5.5 (+454%)
ExecKN	86 (-33%)	53 (-59%)	22 (-83%)	1.8 (+8%)	1.8 (+7%)	1.3 (-27%)	1.1 (+13%)	1.9 (+87%)	5.5 (+453%)
QueueWait	81 (-37%)	77 (-40%)	126 (-1%)	1.6 (-3%)	2.4 (+42%)	5.0 (+183%)	1.5 (+46%)	1.6 (+56%)	0.9 (-6%)

Table 5: Charged cost, cost efficiency and utility for policies on DAS4.

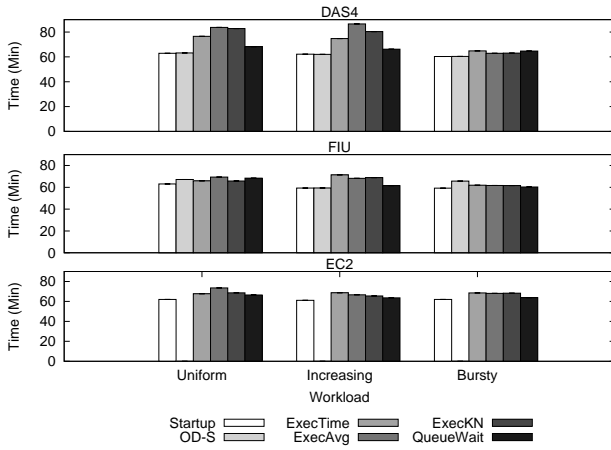


Figure 2: Workload Makespan (WMS).

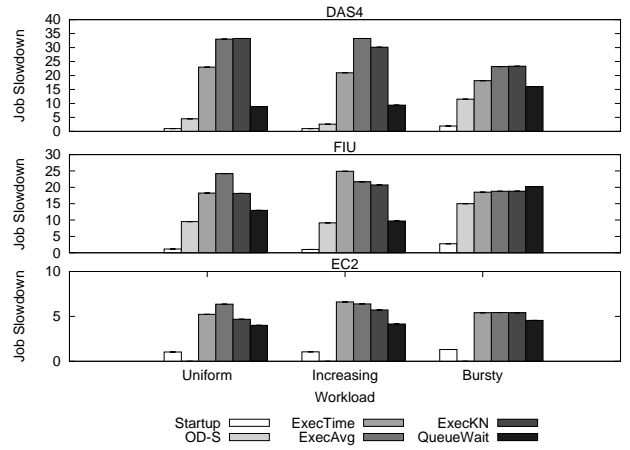


Figure 3: Job Slowdown (JSD).

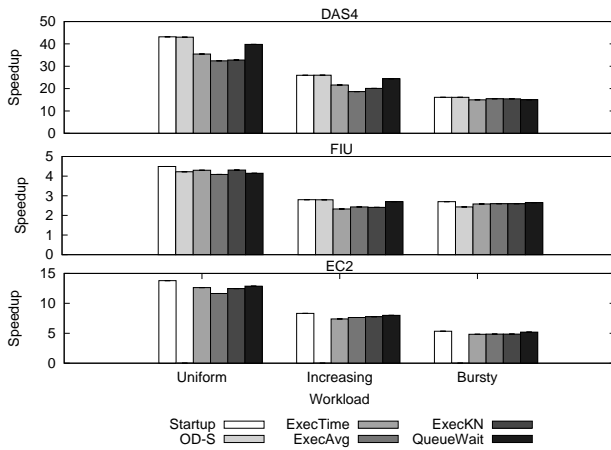
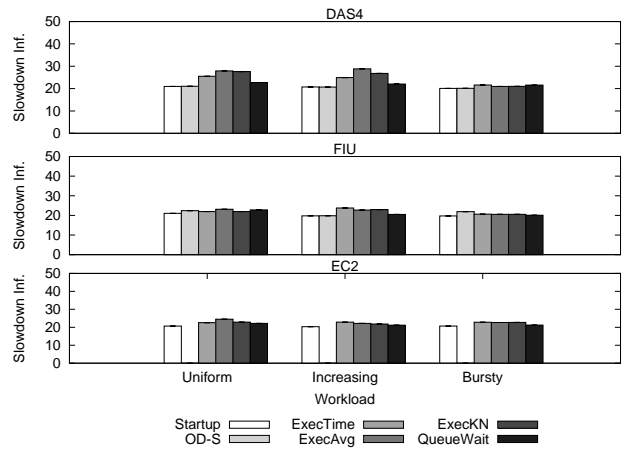


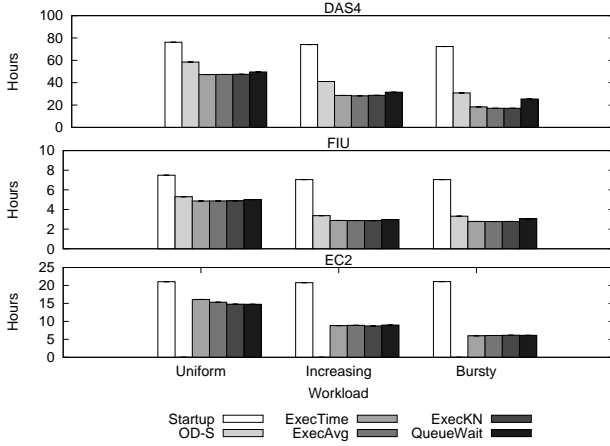
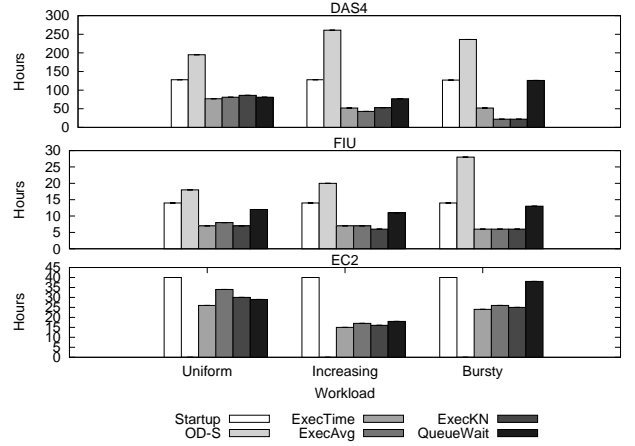
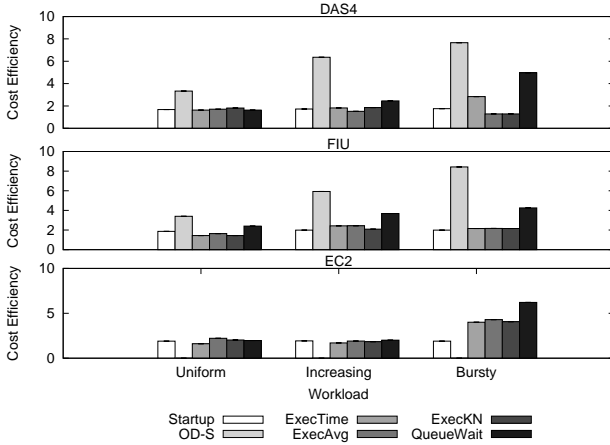
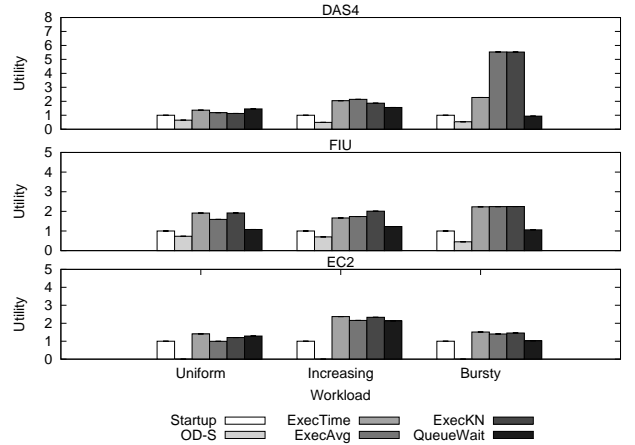
Figure 4: Workload speedup (SU).


 Figure 5: Workload Slowdown Inf. (SU_{∞}).

5.2 Allocation Policies for Static Resources

In this experiment we want to study the performance of different allocation policies and the static provisioning policy, **Startup**. Resources are acquired at the beginning of the experiment, and then jobs are sent to the system.

We use the **FCFS**, the **SJF**, and the **FCFS-NW** allocation policies in the three testbeds. Figure 13 lists the results only for the job slowdown metric, since we did not observe significant differences in cost or makespan. The experiment shows that **SJF** gives a lower slowdown, since shorter jobs are processed first, which means jobs in wait less time in the queue. Overall, **FCFS** performs similarly to **SJF** for the uniform and increasing workloads, however its performance degrades when under a bursty load. Lastly, the **FCFS-NW** policy, which assigns jobs to VMs with round-robin, creates resource competition, and thus has worse results in all experiments.


 Figure 6: Actual cost (C_a).

 Figure 7: Charged cost (C_c).

 Figure 8: Cost efficiency (C_{eff}).

 Figure 9: Utility (U).

5.3 Effects of job size distribution in on-demand policies

In this section, we investigate the impact of the job size distribution. To this end, we create an artificial workload with periodic arrival intervals with different ratios of short and long jobs. For this case, jobs have runtime averages of 10 seconds for short ones and 1 hour for long ones. We consider workloads composed of 25%, 50%, and 75% short jobs (SR25%, SR50%, and SR75%, respectively). The arrival times are distributed in nine groups of 20 jobs each, of 300, 100, 30, 10, 5, 10, 30, 100, and 300 seconds of interarrival time. There are four repetitions of this distribution, resulting in a workload of 720 jobs. Figure 14 shows the interarrival time between jobs over the workload's span. The provisioning policies are able to instantiate up to 50 VMs. The OD-S stops VMs after they have been idle for 10 minutes, while the OD-2Q uses 10 minutes for VMs in the short job pool and 30 seconds in the long job pool. This is set so that short jobs can reuse VMs more often while VMs for long jobs are started only when needed. The number of VMs in the short and long job pools is variable and depends on the demands at a given point in time.

Figure 15 shows that, compared to OD-S with FCFS and OD-S with SJF, the new combination of policies

	ChargedCost			CostEfficiency			Utility		
	Uniform	Increasing	Bursty	Uniform	Increasing	Bursty	Uniform	Increasing	Bursty
Startup	14	14	14	1.9	2.0	2.0	1.0	1.0	1.0
ExecTime	7 (-50%)	7 (-50%)	6 (-57%)	1.4 (-23%)	2.4 (+22%)	2.2 (+9%)	1.9 (+91%)	1.7 (+66%)	2.2 (+123%)
ExecAvg	8 (-43%)	7 (-50%)	6 (-57%)	1.6 (-12%)	2.4 (+23%)	2.2 (+9%)	1.6 (+59%)	1.7 (+74%)	2.2 (+124%)
ExecKN	7 (-50%)	6 (-57%)	6 (-57%)	1.4 (-23%)	2.1 (+5%)	2.2 (+8%)	1.9 (+92%)	2.0 (+101%)	2.2 (+124%)
QueueWait	12 (-14%)	11 (-21%)	13 (-7%)	2.4 (+28%)	3.7 (+85%)	4.3 (+114%)	1.1 (+8%)	1.2 (+23%)	1.1 (+6%)

Table 6: Charged cost, cost efficiency and utility for policies on FIU.

Workload	Charged Cost			Cost Efficiency			Utility		
	Uniform	Increasing	Bursty	Uniform	Increasing	Bursty	Uniform	Increasing	Bursty
Startup	40	40	40	1.9	1.9	1.9	1.0	1.0	1.0
ExecTime	26 (-35%)	15 (-62%)	24 (-40%)	1.6 (-15%)	1.7 (-12%)	4.0 (+111%)	1.4 (+41%)	2.4 (+137%)	1.5 (+51%)
ExecAvg	34 (-15%)	17 (-57%)	26 (-35%)	2.2 (+17%)	1.9 (-1%)	4.3 (+126%)	1.0 (-1%)	2.2 (+116%)	1.4 (+40%)
ExecKN	30 (-25%)	16 (-60%)	25 (-38%)	2.0 (+7%)	1.8 (-5%)	4.1 (+115%)	1.2 (+20%)	2.3 (+133%)	1.5 (+45%)
QueueWait	29 (-28%)	18 (-55%)	38 (-5%)	2.0 (+3%)	2.0 (+4%)	6.2 (+227%)	1.3 (+29%)	2.1 (+113%)	1.0 (+2%)

Table 7: Charged cost, cost efficiency and utility for policies on EC2.

achieves a lower cost for the workload. Also, the slowdown is reduced for workloads with a high number of short jobs. The reason for this is that long jobs are no longer kept in the queue for long time. However, we can see that for workloads with a low number of short jobs, the new policies result in higher slowdowns. This is due to the fact that the workload cannot take advantage of differentiating the two types of jobs, and additionally partitioning the pool of VMs increases the possibility of jobs from one group not having enough resources because the VMs are used by the other jobs, while other policies have a common pool of resources that can be used by any job.

5.4 Policy interactions

We study here the interactions between allocation and provisioning policies. We use the simulator to test six pairs of policies that comprise three provisioning, **Startup**, **OD-S**, and **OD-G**; and two allocation policies, **FCFS** and **SJF**. The **Startup** policy uses 100 VMs during the whole workload, while the on-demand ones can instantiate up to that number of resources. **OD-S** provisions single VMs based on job arrivals, while **OD-G** increases the number by a factor of 2 every time there are no available VMs, and halves the factor when at least one VM has been idle for the pre-defined amount of time, which for both on-demand policies is 10 minutes.

We use three workloads with one thousand jobs where half of the jobs have a runtime average of 10 seconds and the rest of one hour. Jobs of the first workload, Uniform, have an interarrival time of 10 seconds. The second one, Periodic, has four periods of increasing and decreasing arrival times starting and ending at 60 seconds and peaking at 5 seconds. Finally the last one, Bursty, alternates five periods of 200 jobs with high (30 seconds) and low (2 seconds) interarrival times. Figure 16 shows the interarrival distributions for these workloads.

Figure 17 shows the results of this experiment. We did not include makespan in the figure, because all policies had similar values. The top half of Figure 17 shows the cost of running the workloads, and it can be seen how on-demand provisioning policies are much more sensitive to variations in comparison to **Startup**. This is especially noticeable for the periodic workload, which has the highest variability; For the uniform workload, the system is at full utilization most of the time, minimizing the benefits of dynamic provisioning policies. The bottom half of the Figure 17 shows the average job slowdown, and it illustrates how the **SJF** allocation policy reduces the overall overhead for jobs by executing the shorter jobs first, and therefore minimizing the

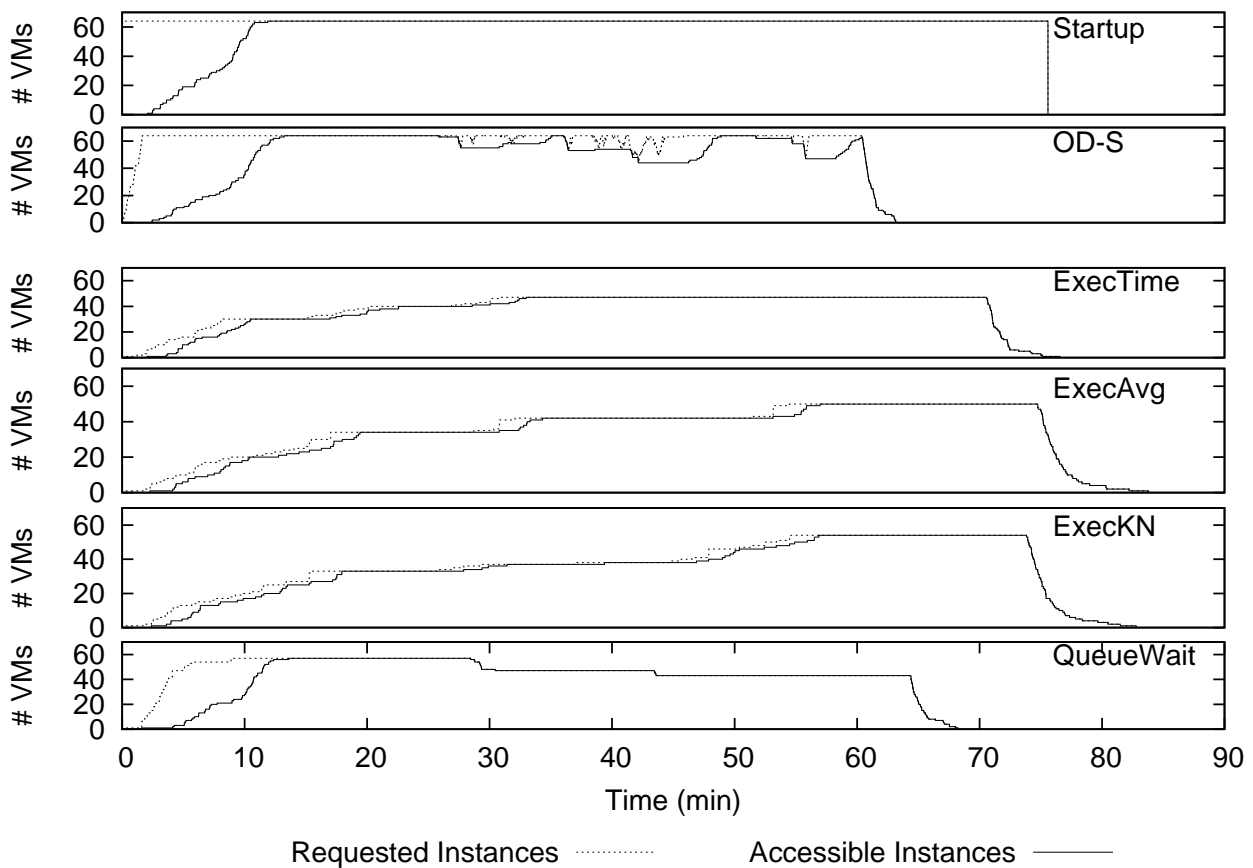


Figure 10: Instances over time for the provisioning policies with the *Uniform* workload on DAS4.

time that jobs wait in general. Another conclusion is that **Startup** results in lower slowdown in comparison to dynamic provisioning policies, due to the lack of overhead for VM booting and shutdown. Additionally, the figure illustrates how the **OD-G** policy results in slightly higher slowdown. The reason for this is that the geometric increase of VMs needs some time to ramp up to reach the required number of VMs to run all jobs in the queue, while the **OD-S** policy instantiates one VM for each waiting job. More detailed information about the rate of VM instantiation and release can be seen in Figure 18.

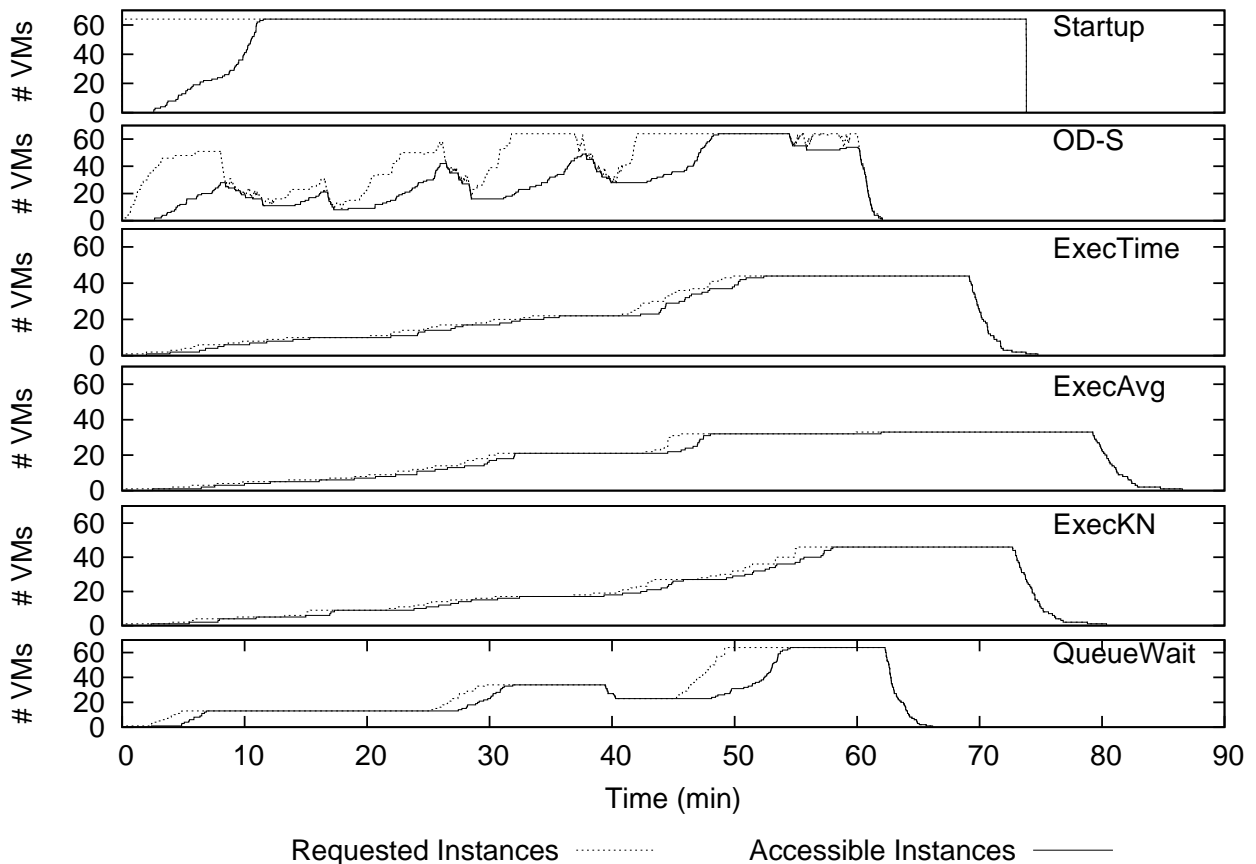


Figure 11: Instances over time for the provisioning policies with the *Increasing* workload on DAS4.

6 Related Work

Much effort [4, 9, 3] has been recently put in investigating algorithms and policies for scheduling jobs on dynamic resources such as clouds. In contrast to our work, which focuses on a system model where all resources are provided from IaaS clouds, most of the related work [21, 22, 23, 24] considers the use of cloud resources only as an extension to an existing, non-cloud system. Moreover, ours is the first comprehensive investigation of provisioning and allocation policies conducted in *real* IaaS clouds.

Closest to our work, Genaud et al. [4] *simulate* various algorithms to provision VMs and assign jobs to them, but consider different policies than in this work and it is unclear if their simulation results are realistic. Assuncao et al. [3] study in simulation three allocation policies and different provisioning policies for extending the capacity of a local private infrastructure with public cloud resources. Murphy et al. [25] discuss an algorithm to provision clusters of VMs for Condor jobs. Kijisiponse et al. [9] consider the FIFO allocation policy and study several VM provisioning policies. In [21], the local site is extended with IaaS resources based on the decision of provisioning policies that react to current load. Lu et al. [22] address the problem of idle instances when executing BLAST workloads by dynamically scaling in Microsoft's Azure cloud. Mao et al. [26] present an algorithm for automatic provisioning to host jobs with deadlines, taking VM startup time and different instance types into consideration. However, they concentrate on the provisioning decisions rather than on job allocation.

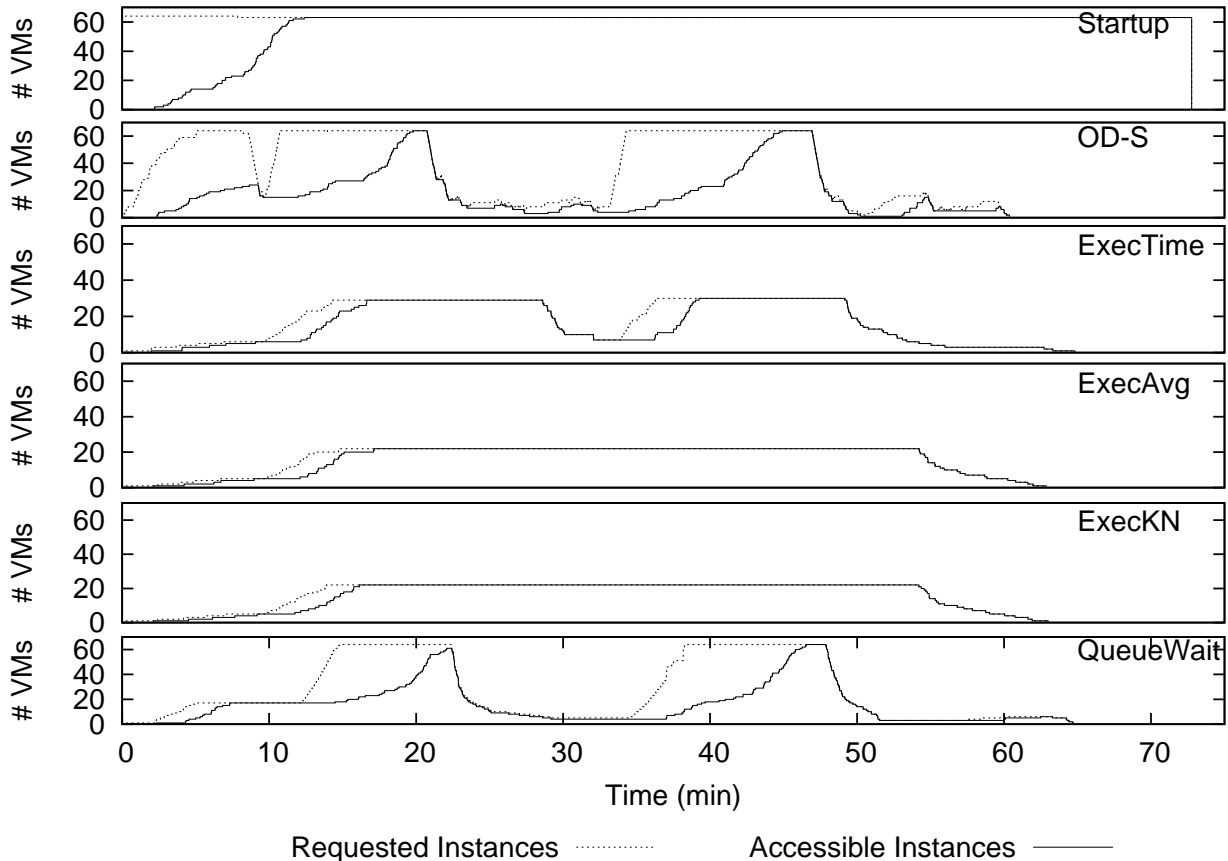


Figure 12: Instances over time for the provisioning policies with the *Bursty* workload on DAS4.

Candeia et al. [23] propose a greedy allocation policy to manage bursts of bags-of-tasks. Ostermann et al. [27] design a hybrid resource manager that dynamically provisions cloud resources and adds them to a grid. Deelman et al. [8] and Wang et al. [28] explore how scientific communities can use cloud resources, where VM instances are leased based on the ratio of waiting jobs to total available VMs.

Other approaches consider complex policies, for example policies [29, 30, 31, 32] that try to predict the workload to determine when to provision new VMs. Cost is another important parameter when provisioning: Henzinger et al. [33] describe a model where a cloud presents different schedules and costs. Other related work [34, 35] uses market approaches to determine when to provision new resources. Our study complements these approaches with a more realistic investigation focusing on simpler policies.

Some authors have considered the effects of decoupling policies: Ranganathan et al. [36] consider job and data scheduling separately to improve data availability. Kee et al. [37] describe an approach where resource discovery and acquisition are integrated. Song et al. [38] perform multi-level scheduling in order to allocate jobs to VMs and VMs to Physical resources. Xu et al. [39] implement a two-level scheduling mechanism to manage resources based on SLAs. Sotomayor et al. [40] consider the overhead impact of transferring a VM to the target host when scheduling a job, and plan such transfers accordingly. [41] discuss the performance of gang scheduling in a cloud, where VMs are acquired on demand. They contemplate the processes of scheduling jobs and handling machines dynamically separately and simulate the behavior of both types of policies. Zhang et

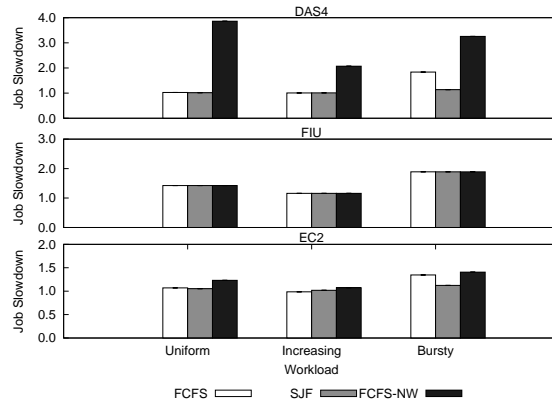


Figure 13: Average Job Slowdown for Allocation policies.

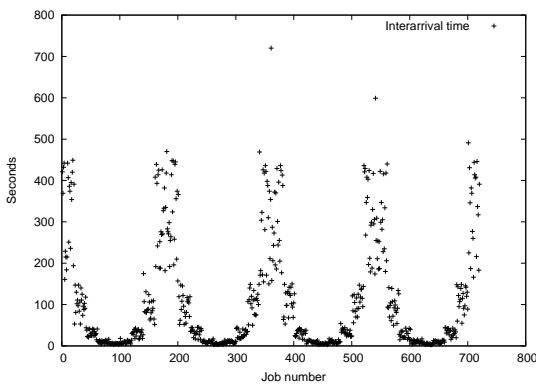


Figure 14: Interarrival time for periodic distribution.

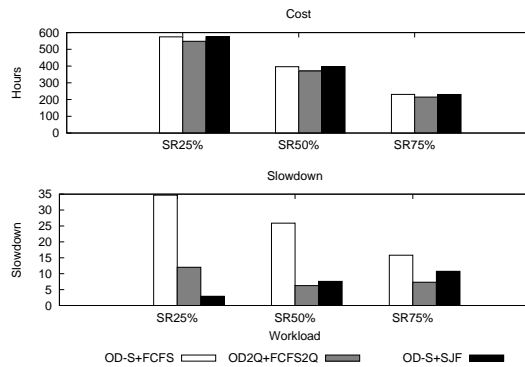


Figure 15: Slowdown and cost for job runtime ratios.

al. [42] integrate resource consumption of a PaaS application and provisioning decisions.

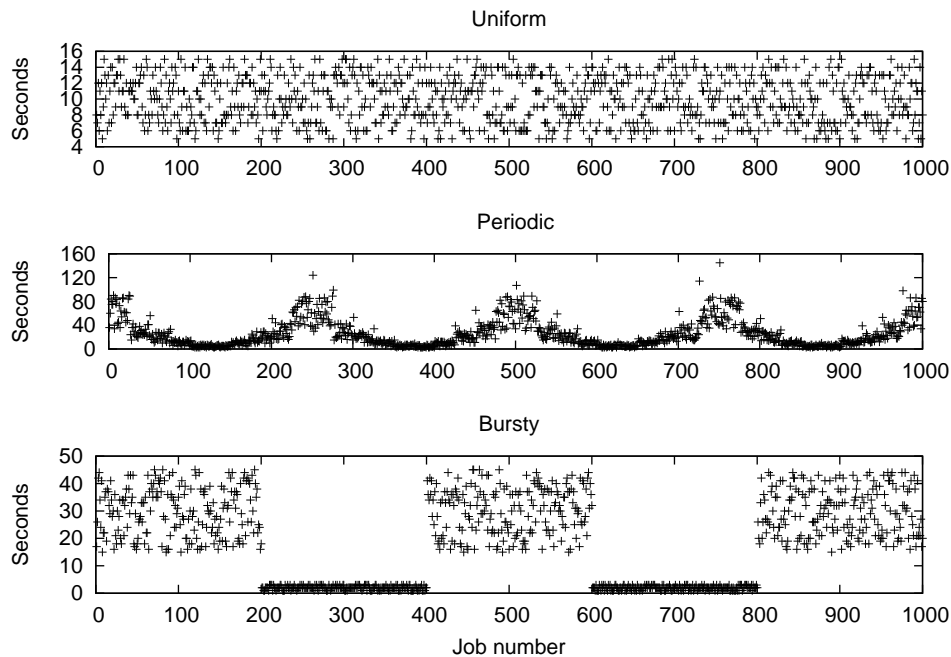


Figure 16: Uniform, Periodic, and Bursty interarrival times.

7 Conclusion and Future Work

To manage their workloads, current and near-future users of IaaS clouds need a better understanding of the performance and cost of provisioning and allocation policies. In this work we have conducted a comprehensive study of these two types of policies, and of the interaction between them.

Overall, we have investigated eight provisioning and four allocation policies, and their interplay. We have developed SkyMark, a framework for IaaS performance evaluation, and conducted with it empirical research in three IaaS clouds, including Amazon EC2. Due to actual resource and budget availability constraints, we have also duplicated SkyMark functionality in a discrete event simulator. Based on results obtained in real and simulated IaaS clouds, we conclude that none of the tested (combined) policies is consistently better than the others across all cases. Our five main findings are summarized below:

1. **OD-ExecTime** and its two variants (especially **OD-ExecKN**), are a good performance-cost trade-off among the investigated provisioning policies;
2. Geometric on-demand provisioning policies, such as **OD-G**, are worse than single-VM on-demand policies;
3. The combined **OD-2Q-FCFS-2Q** policy, which we are the first to investigate in the context of clouds, is a good slowdown-cost trade-off for workloads with a significant ratio of short to long jobs;
4. Naïve static provisioning policies deliver stable performance, but incur up to 5 times higher cost;
5. Allocation policies with information about job runtimes achieve significantly better performance than uninformed allocation policies, when coupled with dynamic provisioning policies.

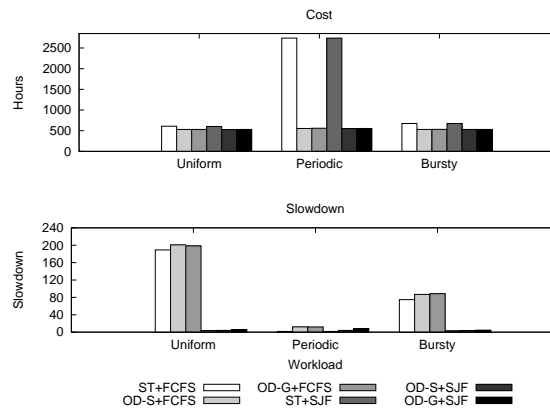


Figure 17: Slowdown and cost for groups of policies.

In the future, we plan to extend this work to consider new provisioning and allocation policies that adapt to changing workload, evolving resources, and complex Service Level Agreements. We will also investigate more diverse types of workloads, such as the typical workloads of grids [14].

Acknowledgments

This material is based upon work supported by the National Science Foundation under Grants No. OISE-0730065 and HRD-083309. This work was also supported by the STW/NWO Veni grant @larGe (11881). The authors would also like to thank Kees Verstoep and Dick Epema, for support with the DAS-4 experiments.

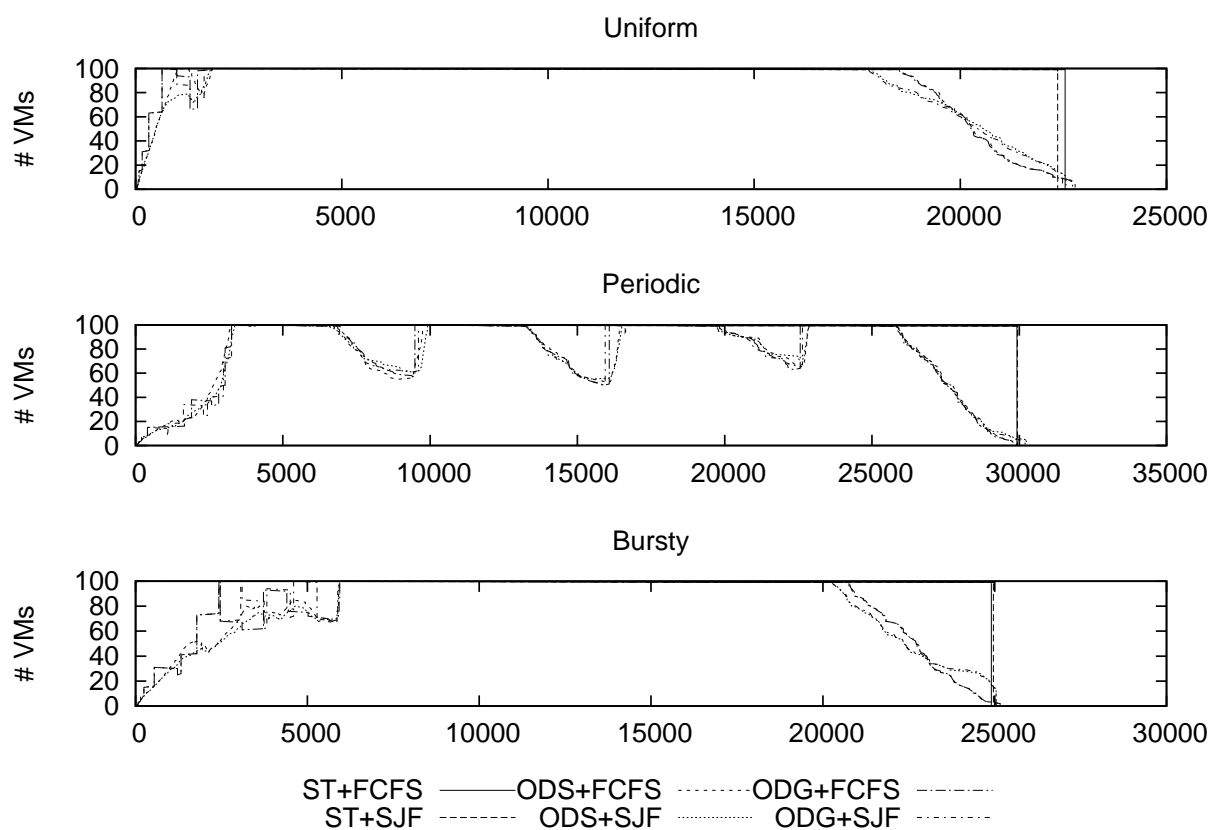


Figure 18: VM provisioning for Uniform, Periodic, and Bursty workloads.

References

- [1] C. Guo, H. Wu, K. Tan, L. Shi, Y. Zhang, and S. Lu, “Dcell: a scalable and fault-tolerant network structure for data centers,” in *SIGCOMM*, 2008, pp. 75–86. [4](#)
- [2] A. G. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, “Vl2: a scalable and flexible data center network,” in *SIGCOMM*, 2009, pp. 51–62. [4](#)
- [3] M. D. de Assuncao, A. di Costanzo, and R. Buyya, “Evaluating the cost-benefit of using cloud computing to extend the capacity of clusters,” in *HPDC*, 2009, pp. 141–150. [4](#), [16](#)
- [4] S. Genaud and J. Gossa, “Cost-wait trade-offs in client-side resource provisioning with elastic clouds,” in *IEEE CLOUD*, july 2011, pp. 1–8. [4](#), [16](#)
- [5] J. Sherwani, N. Ali, N. Lotia, Z. Hayat, and R. Buyya, “Libra: a computational economy-based job scheduling system for clusters,” *Softw., Pract. Exper.*, vol. 34, no. 6, 2004. [4](#)
- [6] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger, “Economic models for resource management and scheduling in grid computing,” *Concurrency and Computation: Practice and Experience*, vol. 14, no. 13-15, pp. 1507–1542, 2002. [4](#)

- [7] D. Thain, T. Tannenbaum, and M. Livny, “Distributed computing in practice: the condor experience,” *Concurrency - Practice and Experience*, vol. 17, no. 2-4, pp. 323–356, 2005. 4
- [8] E. Deelman, G. Singh, M. Livny, G. B. Berriman, and J. Good, “The cost of doing science on the cloud: the montage example,” in *SC*, 2008, p. 50. 4, 17
- [9] E. Kijisipongse and S. Vannarat, “Autonomic resource provisioning in rocks clusters using eucalyptus cloud computing,” in *MEDES*, 2010, pp. 61–66. 4, 16
- [10] D. Jiang, G. Pierre, and C.-H. Chi, “Ec2 performance analysis for resource provisioning of service-oriented applications,” in *ICSOC/ServiceWave Workshops*, 2009, pp. 197–207. 4
- [11] A. Iosup, S. Ostermann, N. Yigitbasi, R. Prodan, T. Fahringer, and D. H. J. Epema, “Performance analysis of cloud computing services for many-tasks scientific computing,” *IEEE TPDS*, vol. 22, no. 6, pp. 931–945, 2011. 4, 5
- [12] A. Iosup, N. Yigitbasi, and D. H. J. Epema, “On the performance variability of production cloud services,” in *CCGRID*, 2011, pp. 104–113. 4
- [13] A. Iosup, H. Li, M. Jan, S. Anoep, C. Dumitrescu, L. Wolters, and D. H. J. Epema, “The grid workloads archive,” *Future Generation Comp. Syst.*, vol. 24, no. 7, pp. 672–686, 2008. 5
- [14] A. Iosup and D. H. J. Epema, “Grid computing workloads,” *IEEE Internet Computing*, vol. 15, no. 2, pp. 19–26, 2011. 5, 20
- [15] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, “Mesos: A platform for fine-grained resource sharing in the data center,” in *NSDI*, 2011. 5
- [16] Y. Chen, A. Ganapathi, R. Griffith, and R. H. Katz, “The case for evaluating mapreduce performance using workload suites,” in *MASCOTS*, 2011, pp. 390–399. 5
- [17] M. A. Iverson, F. Özgüner, and G. J. Follen, “Run-time statistical estimation of task execution times for heterogeneous distributed computing,” in *HPDC*, 1996, pp. 263–. 7
- [18] A. Iosup and D. Epema, “Grenchmark: A framework for analyzing, testing, and comparing grids,” in *CCGRID*, vol. 1, may 2006, pp. 313 – 320. 9
- [19] N. Yigitbasi, A. Iosup, D. Epema, and S. Ostermann, “C-meter: A framework for performance analysis of computing clouds,” in *CCGRID*, may 2009, pp. 472 –477. 9
- [20] A. Iosup, D. H. J. Epema, C. Franke, A. Papaspyrou, L. Schley, B. Song, and R. Yahyapour, “On grid performance evaluation using synthetic workloads,” in *JSSPP*, 2006, pp. 232–255. 10
- [21] P. Marshall, K. Keahey, and T. Freeman, “Elastic site: Using clouds to elastically extend site resources,” *CCGRID*, vol. 0, pp. 43–52, 2010. 16
- [22] W. Lu, J. Jackson, J. Ekanayake, R. Barga, and N. Araujo, “Performing large science experiments on azure: Pitfalls and solutions,” in *CloudCom*, 30 2010-dec. 3 2010, pp. 209 –217. 16
- [23] D. Candeia, R. Araujo, R. Lopes, and F. Brasileiro, “Investigating business-driven cloudburst schedulers for e-science bag-of-tasks applications,” in *CloudCom*, 2010, pp. 343 –350. 16, 17
- [24] O. A. Ben-Yehuda, A. Schuster, A. Sharov, M. Silberstein, and A. Iosup, “ExPERT: Pareto-efficient task replication on grids and clouds,” Technion CS Dept., Tech. Rep. CS-2011-03, Apr 2011. [Online]. Available: <http://www.cs.technion.ac.il/users/wwwb/cgi-bin/tr-info.cgi/2011/CS/CS-2011-03> 16

- [25] M. Murphy, B. Kagey, M. Fenn, and S. Goasguen, “Dynamic provisioning of virtual organization clusters,” in *CCGRID*, may 2009, pp. 364–371. [16](#)
- [26] M. Mao, J. Li, and M. Humphrey, “Cloud auto-scaling with deadline and budget constraints,” in *Grid Computing (GRID), 2010 11th IEEE/ACM International Conference on*, oct. 2010. [16](#)
- [27] S. Ostermann, R. Prodan, and T. Fahringer, “Resource management for hybrid grid and cloud computing,” in *Cloud Computing*, N. Antonopoulos and L. Gillam, Eds., 2010. [17](#)
- [28] L. Wang, J. Zhan, and W. Shi, “In cloud, can scientific communities benefit from the economies of scale?” *TPDS*, vol. PP, no. 99, p. 1, 2011. [17](#)
- [29] A. Quiroz, H. Kim, M. Parashar, N. Gnanasambandam, and N. Sharma, “Towards autonomic workload provisioning for enterprise grids and clouds,” in *Grid*, 2009. [17](#)
- [30] J. Tirado, D. Higuero, F. Isaila, and J. Carretero, “Predictive data grouping and placement for cloud-based elastic server infrastructures,” in *CCGRID*, may 2011, pp. 285–294. [17](#)
- [31] T. J. Hacker and K. Mahadik, “Flexible resource allocation for reliable virtual cluster computing systems,” in *SC*, 2011. [17](#)
- [32] E.-K. Byun, Y.-S. Kee, J.-S. Kim, and S. Maeng, “Cost optimized provisioning of elastic resources for application workflows,” *FGCS*, vol. 27, no. 8, pp. 1011–1026, 2011. [17](#)
- [33] T. Henzinger, A. Singh, V. Singh, T. Wies, and D. Zufferey, “Flexprice: Flexible provisioning of resources in a cloud environment,” in *IEEE CLOUD*, july 2010, pp. 83–90. [17](#)
- [34] L. Wu, S. Garg, and R. Buyya, “Sla-based resource allocation for software as a service provider (saas) in cloud computing environments,” in *CCGRID*, may 2011, pp. 195–204. [17](#)
- [35] M. Salehi and R. Buyya, “Adapting market-oriented scheduling policies for cloud computing,” in *ICA3PP*, 2010. [17](#)
- [36] K. Ranganathan and I. Foster, “Decoupling computation and data scheduling in distributed data-intensive applications,” in *HPDC*, 2002, pp. 352–358. [17](#)
- [37] Y.-S. Kee, K. Yocum, A. A. Chien, and H. Casanova, “Improving grid resource allocation via integrated selection and binding,” in *SC*, 2006. [17](#)
- [38] Y. Song, H. Wang, Y. Li, B. Feng, and Y. Sun, “Multi-tiered on-demand resource scheduling for vm-based data center,” in *CCGRID*, may 2009, pp. 148–155. [17](#)
- [39] J. Xu, M. Zhao, J. Fortes, R. Carpenter, and M. Yousif, “On the use of fuzzy modeling in virtualized data center management,” in *ICAC*, june 2007, p. 25. [17](#)
- [40] B. Sotomayor, K. Keahey, and I. Foster, “Overhead matters: A model for virtual resource management,” in *VTDC*, nov. 2006, p. 5. [17](#)
- [41] I. Moschakis and H. Karatza, “Evaluation of gang scheduling performance and cost in a cloud computing system,” *The Journal of Supercomputing*, pp. 1–18, 2010. [17](#)
- [42] Y. Zhang, G. Huang, X. Liu, and H. Mei, “Integrating resource consumption and allocation for infrastructure resources on-demand,” in *IEEE CLOUD*, july 2010, pp. 75–82. [18](#)