HCloud: Resource-Efficient Provisioning in Shared Cloud Systems

Christina Delimitrou
Stanford University & Cornell University
cdel@stanford.edu

Christos Kozyrakis
Stanford University & EPFL
kozyraki@stanford.edu

Abstract

Cloud computing promises flexibility and high performance for users and cost efficiency for operators. To achieve this, cloud providers offer instances of different sizes, both as long-term reservations and short-term, on-demand allocations. Unfortunately, determining the best provisioning strategy is a complex, multi-dimensional problem that depends on the load fluctuation and duration of incoming jobs, and the performance unpredictability and cost of resources.

We first compare the two main provisioning strategies (reserved and on-demand resources) on Google Compute Engine (GCE) using three representative workload scenarios with batch and latency-critical applications. We show that either approach is suboptimal for performance or cost. We then present HCloud, a hybrid provisioning system that uses both reserved and on-demand resources. HCloud determines which jobs should be mapped to reserved versus on-demand resources based on overall load, and resource unpredictability. It also determines the optimal instance size an application needs to satisfy its Quality of Service (QoS) constraints. We demonstrate that hybrid configurations improve performance by 2.1x compared to fully on-demand provisioning, and reduce cost by 46% compared to fully reserved systems. We also show that hybrid strategies are robust to variation in system and job parameters, such as cost and system load.

Categories and Subject Descriptors: Computer systems organization [*Distributed architectures*]: Cloud computing

Keywords: datacenter; provisioning; QoS; latency; resource efficiency; hybrid; cloud computing

ASPLOS '16, April 02 - 06, 2016, Atlanta, GA, USA

ISBN 978-1-4503-4091-5/16/04…$15.00
DOI: http://dx.doi.org/10.1145/2872362.2872365

1. Introduction

An increasing amount of computing is now hosted in public clouds, such as Amazon's EC2 [1], Windows Azure [65] and Google Compute Engine [24]. Cloud platforms provide two major advantages for end-users and cloud operators: *flexibility* and *cost efficiency* [7,8,29]. Users can quickly launch jobs without the overhead of setting up a new infrastructure every time. Cloud operators can achieve economies of scale by building large-scale datacenters (DCs) and by sharing their resources between multiple users and workloads.

Users need to provision resources along two dimensions; first they must decide between *reserved* and *on-demand* resources. *Reserved resources* are dedicated for long periods of time (typically 1-3 years [1]) and offer consistent service, but come at a significant upfront cost. In the other extreme are *on-demand resources*, which are progressively obtained as they become necessary. The user pays only for resources used at each point, however, the per hour cost is 2-3x higher compared to reserved resources, and acquiring new instances induces instantiation overheads. Second, users must determine the *amount of resources* (instance size) an application needs. While dedicated machines provide more predictable performance they come at a high cost. On the other hand, smaller instances are prone to external interference, resulting in unpredictable quality of service. In this paper we present a provisioning system that navigates these trade-offs.

Since provisioning must determine the necessary resources, it is important to understand the extent of this unpredictability. Performance varies both across instances of the same type (spatial variability), and within a single instance over time (temporal variability) [6, 23, 32, 34, 38, 42, 50, 51, 53, 56, 64]. Figure 1 shows the variability in performance for a Hadoop job running a recommender system using Mahout [41] on various instance types on Amazon EC2 [1] and on Google Compute Engine (GCE) [24]. Analytics such as Hadoop and Spark [67] are throughput-bound applications, therefore performance here corresponds to the completion time of the job. The instances are ordered from smallest to largest, with respect to the number of virtual CPUs and memory allocations they provide. We show 1 vCPU `micro`, 1-8 vCPU standard (`stX`) and 16 vCPU memory-optimized

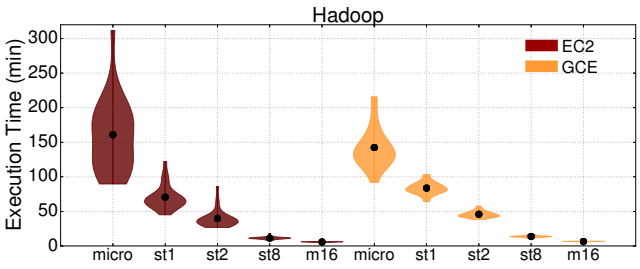


Figure 1: Performance unpredictability on Amazon EC2 and Google Compute Engine for a Hadoop job.

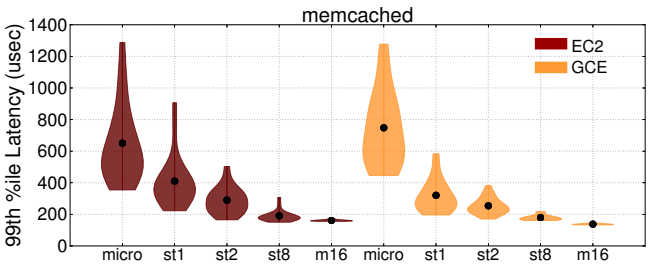


Figure 2: Performance unpredictability on Amazon EC2 and Google Compute Engine for memcached.

Configuration	Cost	Perf. unpredictability	Spin-up	Flexibility	Typical usage
Reserved	High upfront, low per hour	no	no	no	long-term
On-demand	No upfront, high per hour	yes	yes	yes	short-term
Hybrid	Medium upfront, medium per hour	low	some	yes	long-term

Table 1: Comparison of system configurations with respect to: cost, performance unpredictability, overhead and flexibility.

instances (`mX`) [1, 24]. Each graph is the violin plot of completion time of the Hadoop job over 40 instances of the corresponding type. The dot shows the mean performance for each instance type. It becomes clear that especially for instances with less than 8 vCPUs unpredictability is significant, while for the micro instances in EC2 several jobs fail to complete due to the internal EC2 scheduler terminating the VM. For the larger instances (`m16`), performance is more predictable, primarily due to the fact that these instances typically occupy a large fraction of the server, hence they have a much lower probability of suffering from interference from co-scheduled workloads, excluding potential network interference. Between the two cloud providers, EC2 achieves higher average performance than GCE, but exhibits worse tail performance (higher unpredictability).

Figure 2 shows a similar experiment for a latency-critical service (memcached) on the same instance types. Note that the number of memcached clients is scaled by the number of vCPUs of each instance type, to ensure that all instances operate at a similar system load. Unpredictability is even more pronounced now, as memcached needs to satisfy tail latency guarantees [14], as opposed to average performance. The results from above hold, with the smaller instances (less than 8 vCPUs) experiencing significant variability in their tail latency. Performance jitter decreases again for the 8-16 vCPU VMs, especially in the case of the memory-optimized instances (`m16`). Additionally GCE now achieves better average and tail performance compared to EC2.

The goal of this work is to optimize *performance over cost for cloud systems*, similarly to the way work on system design and resource management optimized performance per Watt for small- and large-scale systems [36, 37, 60, 61, 71]. We first explore the implications of the two main provisioning approaches (reserved and on-demand resources), with respect to performance variability and cost efficiency. We perform this analysis on Google Compute Engine (GCE) [24] using three representative workload scenarios with batch and latency-critical applications, and increasing levels of load variability. We assume no a priori knowledge of the applications in each scenario, except for the minimum and maximum aggregate load, which is needed for a comparison with an idealized statically-reserved provisioning strategy. Our study reveals that while reserved resources are superior with respect to performance (2.2x on average over on-demand), they require a long-term commitment, and are therefore suitable for use cases over extended periods of time. Fully on-demand resources, on the other hand, are more cost-efficient for short-term use cases (2.5x on average), but are prone to performance unpredictability, because of instantiation overheads and external interference. We show that to achieve reasonable performance predictability with either strategy, it is crucial to understand the resource preferences and sensitivity to interference of individual applications [21, 43, 54]. Recent work has shown that a combination of lightweight profiling and classification-based analysis can provide accurate estimations of job preferences with respect to the different instance types, the sensitivity to interference in shared resources, and the amount of resources needed to satisfy a job's performance constraints [21].

Next, we design HCloud, a *hybrid provisioning system* that uses both reserved (long-term) and on-demand (short-term) resources. Hybrid provisioning strategies can offer the best of both worlds by leveraging reserved resources for the steady long-term load, and on-demand resources for short-term resource needs. The main challenge with hybrid provisioning is determining how many/what resources to obtain and how to schedule jobs between reserved and on-demand

resources. Table 1 shows the differences between the three provisioning strategies with respect to *cost*, *performance unpredictability*, *instantiation overheads* and *provisioning flexibility*.

We demonstrate that by quickly determining the resource preferences of new jobs, and accounting for system load and instance characteristics, hybrid provisioning strategies improve both resource efficiency and QoS-awareness. They maximize the usage of the already-provisioned reserved resources, while ensuring that applications that can tolerate some performance unpredictability will not delay the scheduling of interference-sensitive workloads. We also compare the performance, cost and provisioning needs of hybrid systems against fully reserved and fully on-demand strategies over a wide spectrum of workload scenarios. Hybrid strategies achieve within 8% of the performance of fully reserved systems (and 2.1x better than on-demand systems), while improving their cost efficiency by 46%. Reserved resources are utilized at 80% on average in steady-state. Finally, we perform a detailed sensitivity analysis of performance and cost with job parameters, such as duration, and system parameters, such as pricing, and load.

2. Cloud Workloads and Systems

2.1 Workload Scenarios

We examine the three scenarios shown in Figure 3 and summarized in Table 2. Each scenario consists of a mix of batch (Hadoop workloads running over Mahout [41] and Spark jobs) and latency-critical workloads (memcached). This mix corresponds to what most online services consist of today, by servicing user requests and analyzing data in the background. The batch jobs are machine learning and data mining applications, including recommender systems, support vector machines, and matrix factorization. memcached is driven with loads that differ with respect to the read:write request ratio, the size of requests, the inter-arrival time distribution, the client fanout and the dataset size.

The first scenario has minimal load variability (*Static*). In steady-state the aggregate resource requirements are 854 cores. Approximately 55% of cores service batch jobs and the remaining 45% latency-critical services. The difference between maximum and minimum load is 10% and most jobs last several minutes to a few tens of minutes.

Second, we examine a scenario with mild, long-term load variability (*Low Variability*). The steady-state minimum load requires on average 605 cores, while in the middle of the scenario the load increases to 900 cores. The surge is mostly caused by an increase in the load of the latency-critical applications. On average 55% of cores are needed for batch jobs and the remaining 45% for the latency-critical services.

Finally, we examine a scenario with large, short-term load changes (*High Variability*). The minimum load drops at 210 cores, while the maximum load reaches up to 1226 cores for short time periods. Approximately 60% of cores are needed for batch jobs and 40% for the latency-critical services. Because of the increased load variability, each individual job is shorter (8.1 min duration on average).

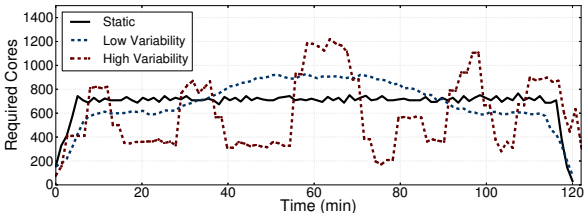


Figure 3: The three workload scenarios.

	Workload Scenarios		
	Static	**Low Var**	**High Var**
max:min resources ratio	1.1x	1.5x	6.2x
batch:low-latency – in jobs	4.2x	3.6x	4.1x
– in cores	1.4x	1.4x	1.5x
inter-arrival times (sec)	1.0	1.0	1.0
ideal completion time (hr)	2.1	2.0	2.0

Table 2: Workload scenario characteristics.

The ideal duration for each scenario, with no scheduling delays or degradation due to interference, is 2 hours.

2.2 Cloud Instances

We use servers on Google Compute Engine (GCE) for all experiments. For provisioning strategies that require smaller instances we start with the largest instances (16 vCPUs) and partition them using Linux containers [5, 13]. The reason for constructing smaller instances as server slices as opposed to directly requesting various instance types is to introduce controlled external interference which corresponds to typical load patterns seen in cloud environments, rather than the random interference patterns present at the specific time we ran each experiment. This ensures *repeatable experiments* and *consistent comparisons* between provisioning strategies.

We model interference by imposing external load that fluctuates $\pm 10\%$ around a 25% utilization [8, 21]. The load is generated by both batch and latency-critical workloads. Section 5.1 includes a sensitivity study to the external load.

We only partition servers at the granularity of existing GCE instances, e.g., 1,2,4,8 and 16 vCPUs. Whenever we refer to the cost of an on-demand instance, we quote the cost of the instance that would be used in the real environment, e.g., a 2 vCPU instance. Similarly, we account for the spin-up overhead of the instance of the desired size, wherever applicable. Finally, all scheduling actions, such as autoscale and VM migration performed by GCE are disabled.

2.3 Cloud Pricing

Google Compute Engine currently only offers on-demand instances. To encourage high instance usage, it provides sustained usage monthly discounts [24]. Although discounts

reduce the prices of on-demand instances, they do not approximate the price of long-term reserved resources. The most popular alternative pricing model is the one used by AWS, which includes both long-term resource reservations and short-term on-demand instances. Because this pricing model offers more provisioning flexibility and captures the largest fraction of the cloud market today, we use it to evaluate the different provisioning strategies. Specifically, we approximate the cost of reserved resources on GCE based on the reserved to on-demand price ratio for EC2, adjusted to the prices of GCE. In Section 5.3 we discuss how our results translate to different pricing models, such as the default GCE model and the pricing model used by Windows Azure.

3. Provisioning Strategies

The two main cloud resource offerings are *reserved* and *on-demand* resources. Reserved instances require a high upfront investment, but have 2-3x lower per-hour cost than on-demand resources, offer better availability, and consistent performance. On-demand resources are charged in a pay-as-you-go manner, but incur spin-up overheads and performance unpredictability due to external load. We ignore spot instances for this work, since they do not provide any availability guarantees. Both with reserved and on-demand resources, the user must also determine the size and number of acquired instances.

Ideally, a provisioning strategy achieves three goals: (1) *high workload performance*, (2) *high resource utilization (minimal overprovisioning)*, and (3) *minimal provisioning and scheduling overheads*. We initially study the three leftmost provisioning strategies described in Table 3.

3.1 Statically Reserved Resources (SR)

This strategy statically provisions reserved resources for 1 year, the shortest contract for reserved resources on EC2. Reserved resources are readily available as jobs arrive, eliminating the overhead of spinning up new VMs. Because SR only reserves large instances (16 vCPUs), there is limited interference from external load, except potentially for some network interference. Because of its static nature, SR must provision resources for the peak requirements of each scenario, plus a small amount of overprovisioning. Overprovisioning is needed because all scenarios contain latency-critical jobs that experience tail latency spikes when using nearly-saturated resources [7, 8, 14, 35]. We explain the insight behind the amount of overprovisioning in Section 3.3. Peak requirements can be easily estimated for static scenarios. For scenarios with load variability, static provisioning results in significant resource underutilization.

3.2 Dynamic On-Demand Resources (OdF, OdM)

We now examine two strategies that acquire resources as they become necessary. On-demand Full (OdF) only uses large instances (16 vCPUs), which are less prone to external interference (see Section 1). On-demand Mixed (OdM) acquires instances of any size, including smaller instances with 1-8 vCPUs. While OdM offers more flexibility, it suffers from substantial external interference in the smaller instances. There are ways to improve predictability in on-demand provisioning strategies, e.g., by sampling multiple instances for each required instance and only keeping the well-behaved ones [23]. Although this addresses performance variability across instances, it is still prone to temporal variation within a single instance. Additionally, it is only beneficial for long-running jobs that can afford the overhead of sampling multiple instances. Short jobs, such as real-time analytics (100msec-10sec) cannot tolerate long scheduling delays and must rely on the initial resource assignment.

	SR	OdF	OdM	HF	HM
Reserved resources	Yes	No	No	Yes	Yes
On-demand resources	No	Yes (full servers)	Yes	Yes (full servers)	Yes

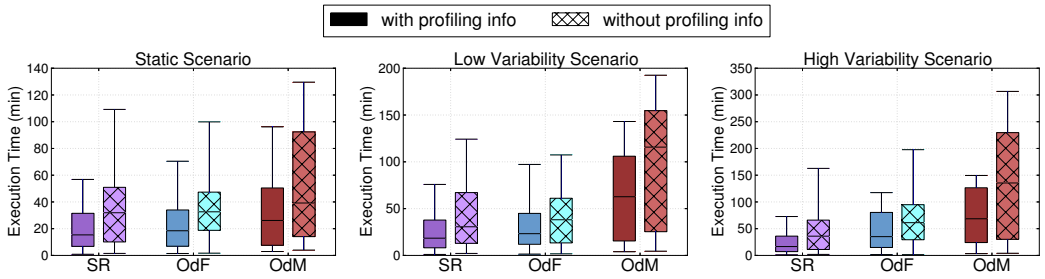
Table 3: Resource provisioning strategies.

Moreover, each new instance incurs overheads to spin up the new VMs. This is typically 12-19 seconds for GCE, although the 95^{th} percentile of spin-up overheads is 2 minutes. Smaller instances tend to incur higher overheads. Therefore provisioning strategies must also decide how long to retain resources after a job completes. If a workload scenario has no or little load variability, instances should be retained to amortize spin-up overheads. We determine retention time by drawing from work on processor power management. The challenge there is to determine when to switch to low power modes that enable power savings but incur overheads to revert to an active mode [40, 45, 59]. Given that the job inter-arrival time in our scenarios is 1 second, we set the retention time to 10x the spin-up overhead. [1] Section 5.1 shows a sensitivity analysis to retention time. Only instances that provide predictably high performance are retained past the completion of their jobs.

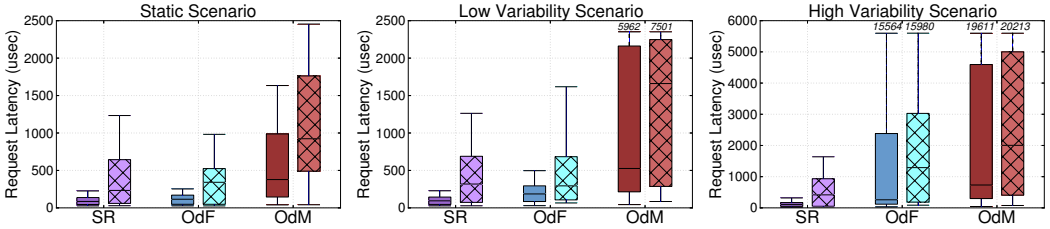
3.3 The Importance of Resource Preferences

Traditionally, end-users have to specify how many resources each job should use; unfortunately this is known to be error-prone and to lead to resource overprovisioning [8, 11, 21, 39, 54]. Moreover, it offers no insight on the sensitivity of each job to interference from other jobs, external or not, running on the same physical server. This is suboptimal for both statically-reserved and on-demand strategies, which may acquire more/fewer resources than needed. SR may colocate jobs that interfere negatively on the same instance. OdF and OdM may acquire instance types that are prone to higher interference than what certain jobs can tolerate.

[1] The benefit of longer retention time varies across instance sizes due to differences in spin-up overheads.



(a) Performance of batch applications for the three scenarios.



(b) Performance of latency-critical applications for the three scenarios.

Figure 4: Performance of jobs of the three scenarios with the three provisioning strategies. The boundaries of the boxplots depict the 25th and 75th percentiles, the whiskers the 5th and 95th, and the horizontal line in each boxplot shows the mean.

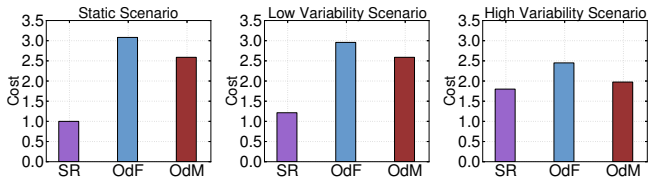


Figure 5: Cost of fully reserved and on-demand systems.

The recently-proposed Quasar cluster manager provides a methodology to quickly determine the resource preferences of new, unknown jobs [21]. When a job is submitted, it is first profiled on two instance types, while injecting interference in two shared resources, e.g., last level cache and network bandwidth. This signal is used by a set of classification techniques which find similarities between the new and previously-scheduled jobs with respect to instance type preferences and sensitivity to interference. A job's sensitivity to interference in resource i is denoted by c_i, where $i \in [1, N]$, and $N = 10$ the number of examined resources [21]. Large c_i values mean that the job puts a lot of pressure in resource i. To capture the fact that certain jobs are more sensitive to specific resources we rearrange vector $C = [c_1, c_2, ..., c_N]$ by order of decreasing magnitude of c_i, $C' = [c_j, c_k, ..., c_n]$. Finally, to obtain a single value for C', we use the order preserving encoding: $Q = c_j \cdot 10^{(2\cdot(N-1))} + c_k \cdot 10^{(2\cdot(N-2))} + ... + c_n$, and normalize Q in $[0, 1]$. Q denotes the resource quality a job needs to satisfy its QoS constraints. High Q denotes a resource-demanding job, while low Q a job that can tolerate some resource interference.

We use Quasar's estimations of resource preferences and interference sensitivity to improve provisioning. For SR, we use these estimations to find the most suitable reserved resources available with respect to size and interference using a simple greedy search [21]. Accounting for resource preferences reduces overprovisioning to 10-15%. For OdF, the estimations are used to select the minimum amount of resources for a job, and to match the resource capabilities of instances to the interference requirements of a job. For OdM, this additionally involves requesting an appropriate instance size and type (standard, compute- or memory-optimized). Note that because smaller instances are prone to external interference, provisioning decisions may have lower accuracy.

Finally, we detect suboptimal application performance and revisit the allocation decisions at runtime [3, 4, 12, 21, 55]. Once a job is scheduled its performance is monitored and compared against its expected QoS. If performance drops below QoS the cluster manager takes action [21]. At a high level, we first try to restore performance through local actions, e.g., increasing the resource allocation on the server where the application already resides, and then through rescheduling. The latter is unlikely in practice.

3.4 Provisioning Strategies Comparison

Performance: We first compare the performance impact of the three provisioning strategies, with and without Quasar's information on job preferences. Figure 4 shows the performance achieved by each of the three strategies for the three workload scenarios. We separate batch (Hadoop, Spark) from latency-critical applications (memcached), since their critical performance metric is different: completion time for

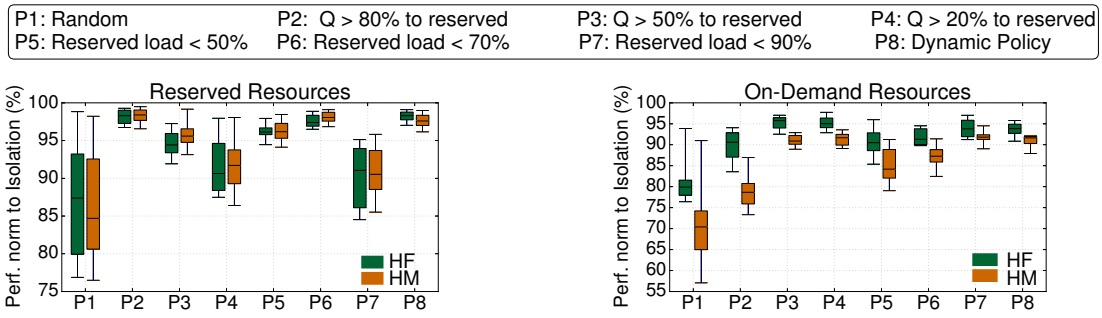


Figure 6: Sensitivity to the policy of mapping jobs to reserved versus on-demand resources for HF and HM.

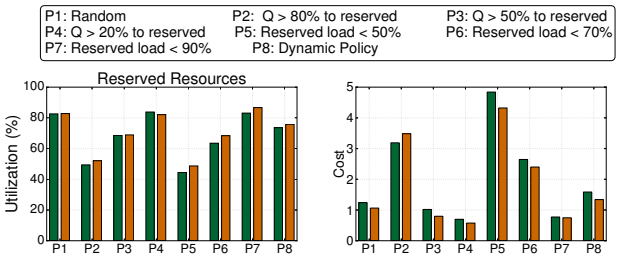


Figure 7: Utilization of reserved resources and cost with different application mapping policies for HF and HM.

the batch jobs and request latency distribution for memcached. The boundaries in each boxplot depict the 25th and 75th percentiles of performance, the whiskers the 5th and 95th percentiles and the horizontal line shows the mean. When Quasar is not used, the resources for each job are sized based on user-defined reservations. For batch jobs (Hadoop and Spark) this translates to using the default framework parameters (e.g., 64KB block size, 1GB heapsize for Hadoop), while for memcached resources are provisioned for peak input load [54]. OdM requests the smallest instance size that satisfies the resource demands of a job. SR allocates resources on the reserved instances with the most available resources (least-loaded).

It is clear from Figure 4 that all three strategies benefit significantly from understanding the jobs' resource preferences and interference sensitivity. Specifically for SR, there is a 2.4x difference in performance on average across scenarios. The differences are even more pronounced for latency-critical jobs, where the performance metric of interest is tail latency. In all following results, we assume that provisioning strategies take job preferences into account, unless otherwise specified.

We now compare the performance achieved by the three provisioning strategies. The static strategy SR achieves the best performance for all three scenarios, both for batch and latency-critical jobs. OdF behaves near-optimally for the static scenario, but worsens for the scenarios where variability is present, primarily due to spin-up overheads to obtain new resources. OdM achieves the worst performance for every scenario (2.2x worse than SR on average), in part because of the spin-up overheads, but primarily because of the performance unpredictability it experiences from external load in the smaller instances. Memcached suffers a 24x and 42x increase in tail latency in the low- and high-variability scenarios, as it is more sensitive to resource interference.

Cost: Figure 5 shows the relative cost of each strategy for the three scenarios. All costs are normalized to the cost of the static scenario with SR. Although strategy SR appears to have the lowest cost for a 2 hour run (2-3x lower per hour charge than on-demand), it requires at least a 1-year commitment with all charges happening in advance. Therefore, unless a user plans to leverage the cluster for long periods of time, on-demand resources are dramatically more cost-efficient. Moreover, SR is not cost effective in the presence of high workload variability, since it results in significant overprovisioning. Between the two on-demand strategies, OdM incurs lower cost, since it uses smaller instances, while OdF only uses the largest instances available. However the cost savings of OdM translate to a significant performance degradation from unpredictability (Figure 4).

4. Hybrid Provisioning Strategies

The previous section showed that neither fully reserved nor fully on-demand strategies are ideal. Hybrid provisioning strategies that combine reserved and on-demand resources have the potential to achieve the best of both worlds. The main challenge now becomes determining which jobs should be scheduled on on-demand versus reserved resources.

4.1 Provisioning Strategies

We design HCloud, a hybrid provisioning system that uses both reserved and on-demand resources, and discuss two strategies HCloud employs. The first strategy (HF) only uses large instances for on-demand resources, to constrain unpredictability. The second strategy (HM), uses a mix of on-demand instance types to reduce cost, including smaller instances that experience interference from external load. The retention time policy for on-demand resources is the same as for the purely on-demand strategies OdF and OdM. The

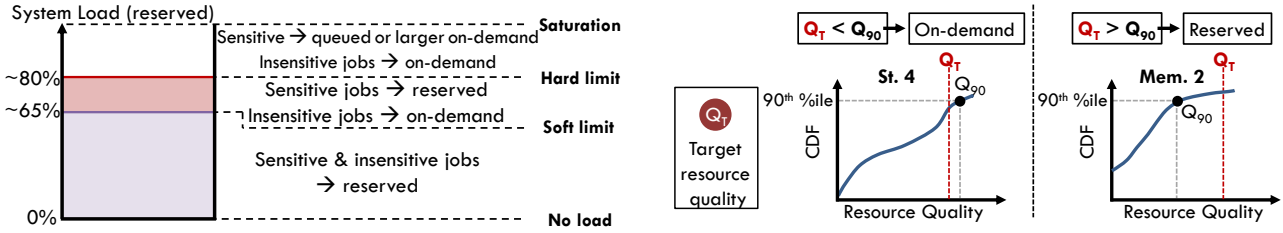


Figure 8: Application mapping scheme between reserved and on-demand instances for HF and HM.

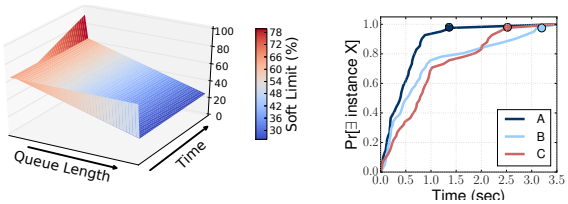


Figure 9: Determining the soft utilization limit (left) and the expected waiting time (right) in HF and HM.

reserved resources are large instances, as with the statically-provisioned strategy. We configure reserved instances to accommodate the minimum steady-state load, e.g., 600 cores for the low variability scenario to avoid overprovisioning. For scenarios with low steady-state load but high load variability the majority of resources are on-demand. Since HF uses large instances for both reserved and on-demand resources, it primarily uses on-demand instances to serve overflow load. In contrast HM can leverage smaller on-demand instances to also accommodate jobs that can tolerate some interference in shared resources.

HCloud addresses two challenges: first, it determines how many resources each incoming job needs to satisfy its QoS, and second, it determines whether a new job should be scheduled on on-demand or reserved resources.

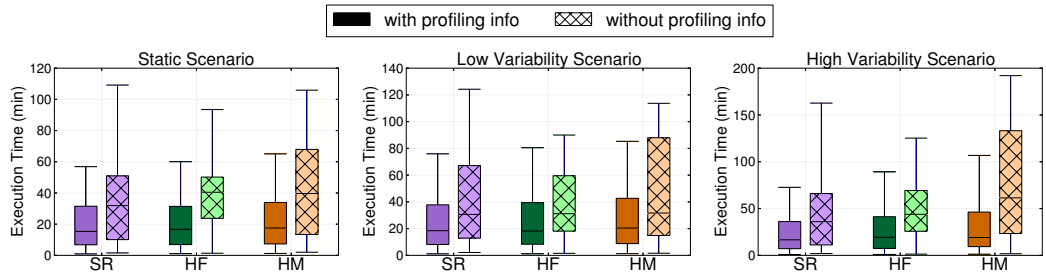
4.2 Application Mapping Policies

We first consider a baseline policy that maps applications between the reserved and on-demand resources randomly using a fair coin. Figure 6 shows the performance of applications mapped to reserved (left) and on-demand resources (right) for the two hybrid strategies in the high variability scenario. Performance is normalized to the performance each job achieves if it runs with unlimited resources in isolation. Figure 7 also shows the utilization of reserved instances and the total cost for the 2 hour scenario normalized to the cost of the static scenario with SR. Because the number of jobs is large, approximately half of them are scheduled on reserved and half on on-demand resources [27]. The random policy hurts performance for jobs mapped to either resource type. In reserved resources, performance degrades as more workloads than the instances can accommodate are assigned to them, and are therefore queued. In on-demand resources, performance degrades for two reasons. First, because on-demand resources introduce instantiation overheads, and, more prominently, because jobs that are sensitive to interference and should have been mapped to reserved resources underperform due to external load.

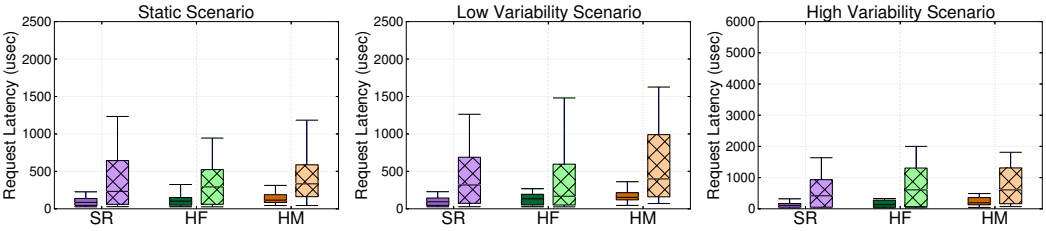
Ideally, the mapping policy should take into account the sensitivity of jobs to performance unpredictability. The following three policies shown in Figure 6 set a limit to the jobs that should be mapped to reserved resources based on the quality of resources they need. $P2$ assigns jobs that need quality $Q > 80\%$ to the reserved instances to protect them from the variability of on-demand resources. $P3$ and $P4$ set stricter limits, with $P4$ only assigning very tolerant to unpredictability jobs to on-demand resources. As we move from $P2$ to $P4$ the performance of jobs in the on-demand instances improves, as the number of jobs mapped to them decreases. In contrast, the performance of jobs scheduled to reserved resources worsens due to increased demand and queueing delays. In general, performance is worse for HM in on-demand resources, due to the increased performance variability of smaller instances.

It is clear that there needs to be an upper load limit for reserved resources. The next three policies $P5 - P7$ set progressively higher, static limits. For low limits, e.g., 50-70% the performance of jobs on reserved resources is near-optimal. In contrast, jobs assigned to on-demand resources suffer substantial degradations, since job mapping is only determined based on load and not resource preferences. For a utilization limit of 90%, the performance of jobs in reserved resources degrades due to excessive load. Low utilization in reserved resources also increases cost, as additional on-demand instances have to be obtained. Therefore a static utilization limit that does not account for resource preferences of jobs is also suboptimal.

Based on these findings we design a dynamic policy to separate jobs between reserved and on-demand resources. The policy adheres to three principles. First, it utilizes reserved resources before resorting to on-demand resources. Second, applications that can be accommodated by on-demand resources should not delay the scheduling of jobs sensitive to interference. Third, the system must adjust the utilization limits of reserved instances to respond to performance degradations due to excessive queueing.



(a) Performance of batch applications for the three scenarios.



(b) Performance of latency-critical applications for the three scenarios.

Figure 10: Performance of the three scenarios with the statically-reserved and hybrid strategies. The boundaries of the boxplots depict the 25th and 75th percentiles, the whiskers the 5th and 95th percentiles and the horizontal line in each boxplot the mean.

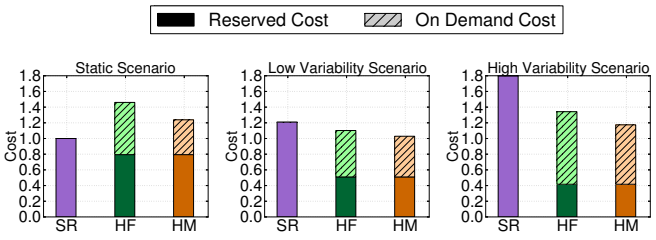


Figure 11: Cost comparison between SR, HF and HM.

Figure 8 explains the dynamic policy. We set two utilization limits for reserved resources. First, a *soft limit* (experimentally set at 60-65% utilization), below which all incoming jobs are allocated reserved resources. Once utilization exceeds this limit, the policy differentiates between jobs that are sensitive to performance unpredictability and insensitive ones. The differentiation is done based on the resource quality Q a job needs to satisfy its QoS constraints and the knowledge on the quality of previously-obtained on-demand instances. Once we determine the instance size a job needs (number of cores, memory and storage), we compare the 90^{th} percentile of quality of that instance type (monitored over time) against the target quality (Q_T) the job needs. If $Q_{90} > Q_T$ the job is scheduled on the on-demand instance, otherwise it is scheduled on the reserved instances. Examining the 90^{th} percentile is sufficient to ensure accurate decisions for the majority of jobs. Tightening this constraint to higher percentiles increases the utilization of reserved resources, and/or the cost from acquiring additional on-demand instances.

Second, we set a *hard limit* for utilization, when jobs need to be queued before reserved resources become available. At this point, any jobs for which on-demand resources are satisfactory are scheduled in on-demand instances and all remaining jobs are locally queued [52]. An exception occurs for jobs whose queueing time would exceed the instantiation overhead of a large on-demand instance (16 vCPUs); these jobs are instead assigned to on-demand instances. Queueing time is estimated using a simple feedback loop based on the rate at which instances of a given type are being released over time. For example, if out of 100 jobs waiting for an instance with 4 vCPUs and 15GB of RAM, 99 were scheduled in less than 1.4 seconds, the system will estimate that there is a 0.99 probability that the queueing time for a job waiting for a 4 vCPU instance will be 1.4 seconds. Figure 9b shows a validation of the estimation of waiting time for three instance types. The lines show the cumulative distribution function (CDF) of the probability that an instance of a given type becomes available. The dots show the estimated queueing time for jobs waiting to be scheduled on instances with 4 (A), 8 (B) and 16 vCPUs (C) in the high variability scenario. In all cases the deviation between estimated and measured queueing time is minimal.

Third, we adjust the soft utilization limit based on the rate at which jobs get queued. If the queued jobs increase sharply, the reserved instances should become more selective in the workloads they accept, i.e., the soft limit should decrease. Similarly, if no jobs get queued for significant periods of time, the soft limit should increase to accept more jobs. We use a simple feedback loop with linear transfer functions to adjust the soft utilization limit of reserved instances as

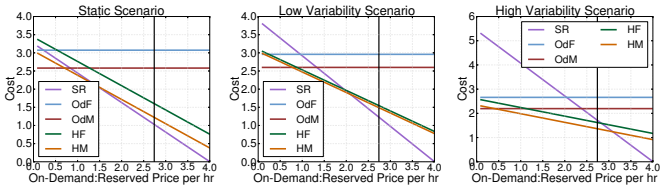


Figure 12: Sensitivity of provisioning cost to on-demand:reserved resource cost.

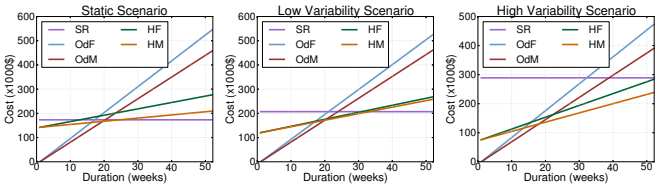


Figure 13: Sensitivity of provisioning cost to workload scenario duration.

a scenario progresses. Figure 9a shows how the soft limit changes with execution time and queue length.

4.3 Provisioning Strategies Comparison

Performance: Figure 10 compares the performance achieved by the static strategy SR and the two hybrid strategies (HF and HM), with and without determining the resource preferences of new jobs. As expected, using the profiling information improves performance significantly for the hybrid strategies, for the additional reason that it is needed to decide which jobs should be scheduled on reserved resources (2.4x improvement on average for HF and 2.77x for HM). When using the profiling information, strategies HF and HM come within 8% of the performance of the statically reserved system (SR), and in most cases outperform strategies OdF and OdM, especially for scenarios with load variability. The main reason is that HF and HM differentiate between jobs that can tolerate the unpredictability of on-demand instances, and jobs that need the predictable performance of a fully controlled environment. Additionally hybrid strategies hide some of the spin-up overhead of on-demand resources by accommodating part of the load in the reserved instances.

Cost: Figure 11 shows the relative cost of strategies SR, HF and HM for the three scenarios. While the static strategy (SR) is more cost-efficient in the static scenario where provisioning is straight-forward, the hybrid strategies incur significantly lower costs for both scenarios with load variability. Therefore, unless load is almost completely static, SR requires high cost, and significant overprovisioning. Additionally, because of the lower per-hour cost of reserved resources in HF and HM, the hybrid strategies have lower per-hour cost than fully on-demand resources as well. For HF and HM, most of the cost per hour comes from on-demand resources, since reserved instances are provisioned for the minimum steady-state load. Finally, between the two hybrid strategies, HM incurs lower cost since it uses smaller instances.

5. HCloud Analysis & Robustness

5.1 Sensitivity to Job & System Parameters

We first evaluate the sensitivity of the previous findings to various system and workload parameters.

Resource cost: The current average cost ratio of on-demand to reserved resources per hour is 2.74. Figure 12 shows how the relative cost of the three scenarios varies when this ratio changes. All costs are normalized to the cost for the static scenario using SR. We change the ratio in $[0.01, 4]$ by scaling the price of reserved resources; beyond that point the cost of SR per hour becomes practically negligible. Initially (0.01), strategies using on-demand resources (OdF, OdM) are significantly more cost-efficient, especially for scenarios with variability. For the static scenario, even when on-demand resources are much cheaper than reserved, SR, HF and HM incur similar charges as the fully on-demand systems. For each scenario, there is a price ratio for which SR becomes the most cost-efficient strategy. As variability increases, this value becomes larger (e.g., for the high variability scenario the ratio must be 3 for SR to be more cost-efficient than HM). Note that SR still requires at least a 1-year commitment, in contrast to on-demand strategies. Finally, the hybrid strategies, HF and HM, achieve the lowest per-hour cost for extended ratio ranges, especially for scenarios with load variability.

Scenario duration: Figure 13 shows the cost of each strategy, as scenario duration increases. Because we compare aggregate costs (instead of per-hour), this figure shows the absolute cost in dollars. For the static scenario, cost-wise, strategy HM is optimal only if duration is $[20 - 25]$ weeks. For durations less than 20 weeks, OdM is the most cost-efficient, while for durations exceeding 25 weeks the statically-reserved system (SR) is optimal. This changes when there is load variability. For the scenario with high variability and durations over 18 weeks, HM is the most cost-efficient strategy, with the significantly overprovisioned reserved system (SR) never being optimal. Note that the charge for SR doubles beyond the 1 year (52 weeks) mark.

Spin-up overhead: Figure 14a shows the 95^{th} percentile of performance for different spin-up overheads in the high variability scenario. The statically-reserved strategy (SR) is not affected by this change. Because in this scenario resources are frequently recycled, increasing the spin-up overheads significantly affects performance. This is more pronounced for strategies using exclusively on-demand resources (OdF, OdM). The additional degradation for OdM comes from the performance unpredictability of smaller instances.

External load: Figure 14b shows the sensitivity of performance to external load (system load due to jobs beyond those

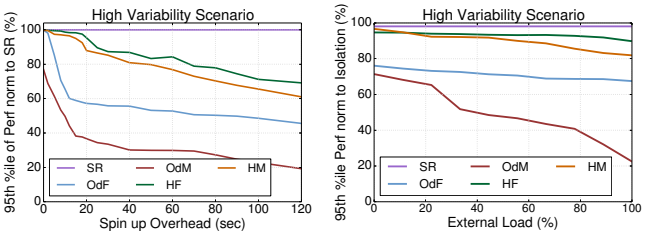


Figure 14: Performance sensitivity to instance spin-up time and external load.

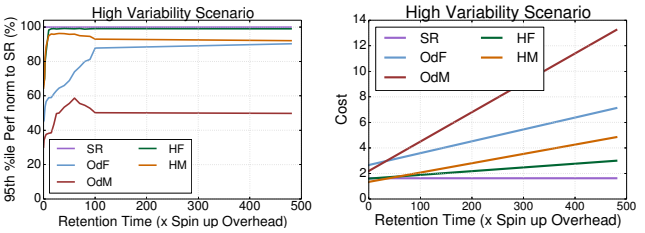


Figure 15: Performance and cost sensitivity to resource retention time.

provisioned with our strategies). SR provisions a reserved system, therefore there is no external load to affect performance. OdF and HF are also tolerant to external load, since they only use large instances, which are less prone to interference. For HM, performance degrades minimally until 50% load, beyond which point the estimations on resource quality become inaccurate. OdM suffers the largest performance degradation since all of its resources are susceptible to external interference.

Retention time: Figure 15 shows the 95^{th} latency percentile and cost for the high variability scenario, as the time for which idle instances are maintained changes. As expected, releasing well-behaved instances immediately hurts performance, since it introduces higher spin-up overheads to acquire new resources. This is especially true here, where load changes frequently. Higher retention time also increases cost for OdF and OdM, while SR remains unchanged; the difference for hybrid strategies is small. An unexpected finding is that excessive resource retention hurts performance slightly for OdM and HM. The primary reason is the temporal variability in the quality of on-demand resources, which degraded by the time new jobs were assigned to them.

Workload characteristics: Figure 16 shows the 95^{th} percentile of performance for the five strategies as the percentage of jobs that are sensitive to interference increases. We modify the high variability scenario used before, such that the number of jobs that cannot tolerate performance unpredictability increases. In the left-most part of the graph, most jobs are batch Hadoop applications, which can tolerate some resource contention; as we move to the right part of the graph

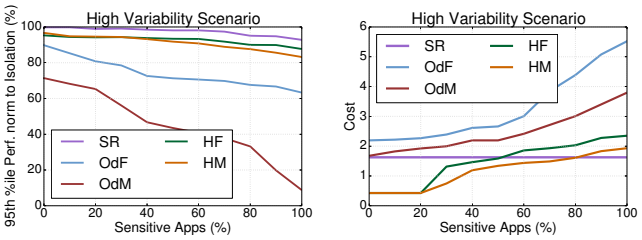


Figure 16: Performance and cost sensitivity to application characteristics.

the majority of jobs are latency-critical memcached applications and real-time Spark jobs.

The statically-provisioned strategy (SR) behaves well even when most applications need resources of high quality, since it is provisioned for peak load, and there is no external load affecting application performance. The two hybrid strategies also behave well, until the fraction of sensitive applications increases beyond 80%, at which point queueing in the reserved resources becomes significant. The purely on-demand strategies are the ones that suffer the most from increasing the fraction of sensitive applications. OdF and especially OdM significantly degrade the performance of scheduled workloads, both due to increased spin-up overheads, and because more applications are now affected by external resource contention.

With respect to cost, increasing the fraction of applications that are sensitive to interference impacts all strategies except for SR. Since HF and HM can use the reserved resources for the sensitive jobs, their cost increases only beyond the 30% mark, at which point more on-demand resources have to be purchased to avoid increased queueing in the reserved resources. The two on-demand strategies experience a significant cost surge, since increasing the fraction of sensitive applications results in a lower degree of coscheduling and makes acquiring new resources necessary.

5.2 Provisioning Overheads

In the presented strategies, the provisioning overheads include job profiling and classification (Quasar), provisioning decisions, spin-up of new on-demand instances (where applicable), and rescheduling actions. The profiling that generates the input signal for classification takes 5-10 sec, but only needs to happen the first time a job is submitted. Classification itself takes 20msec on average. Decision overheads include the greedy scheduler in the static strategy (SR) and the job mapping policy between reserved and on-demand resources in the hybrid strategies. In all cases decision overheads do not exceed 20msec, three orders of magnitude lower than the spin-up overheads of on-demand instances (10-20sec on average). Finally, job rescheduling due to suboptimal performance is very infrequent for all strategies except OdM, where it induces 6.1% overheads to the execution time of jobs on average.

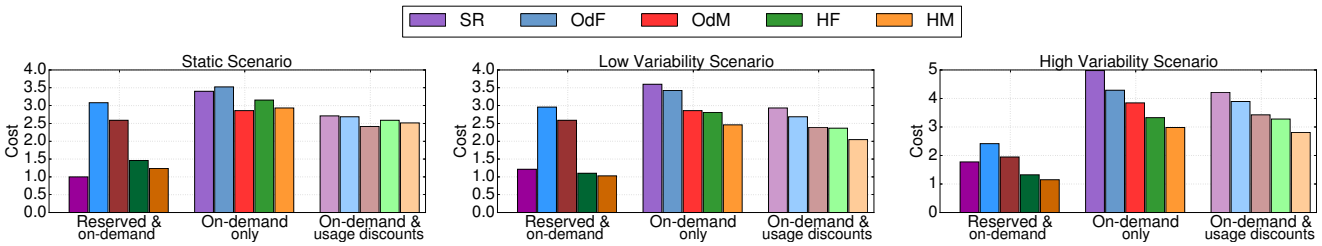


Figure 17: Sensitivity to the cloud pricing model for the three workload scenarios.

5.3 Different Pricing Models

So far we have assumed a pricing model similar to Amazon's AWS. This is a popular approach followed by many smaller cloud providers. Nevertheless, there are alternative approaches. GCE does not offer long-term reservations. Instead it provides *sustained usage monthly discounts* to encourage high utilization of on-demand resources. The higher the usage of a set of instances for a fraction of the month, the lower the per-hour price for the remainder of the month. Microsoft Azure only offers on-demand resources at the moment.

Even without reserved resources, the problem of selecting the appropriate instance size and configuration, and determining how long to retain an instance remains. Figure 17 shows how cost changes for the three workload scenarios, under the Azure (on-demand only) and GCE (on-demand + usage discounts) pricing models, compared to the AWS pricing model (reserved + on-demand). We assume that the resources will be used at least for one month, so that GCE's discounts can take effect. Cost is normalized to the cost of the static scenario under the SR strategy using the *reserved & on-demand* pricing model. Even with the alternative pricing models using the hybrid strategies and accounting for resource preferences of new jobs significantly benefits cost. For example, for the high variability scenario, HM incurs 32% lower cost than OdF with the Windows Azure pricing model; similarly for the GCE model with discounts, HM achieves 30% lower cost than OdF.

GCE decouples the usage from the specific instance used. For example, it does not differentiate between using a single instance of type A for 3 weeks, or 3 instances of type A for 1 week each. This introduces new optimization opportunities to maximize the time a certain instance type is used during a month. We defer such considerations to future work.

5.4 Resource Efficiency

Apart from lowering cost, we must ensure that a provisioning strategy is not wasteful in terms of resources. Figure 18 shows the resource allocation for each strategy throughout the duration of the high variability scenario. SR is provisioned statically for the peak requirements plus a 15% overprovisioning to avoid operating close to saturation, as described in Section 3.1. Because all instances are private (limited external interference) and resources are readily available, SR achieves near-ideal execution time (∼2hr). However, due to the high load variability, system utilization is rarely high, resulting in poor resource efficiency. The majority of resources is only used in the interval between 32 and 60 minutes.

Figure 20 shows the CPU utilization of each instance throughout the execution of the high variability scenario for the five provisioning strategies. CPU utilization is sampled every 2 seconds and averaged across the cores of an instance. For the hybrid strategies we separate the reserved (bottom) from the on-demand resources (top). Servers are ranked from most- to least-utilized at each point in time during the execution of the scenario. Figure 19 also shows server resources in the order in which on-demand instances are obtained. For strategies with only reserved resources, the y-axis corresponds to server IDs.

Strategy OdF obtains resources as they become necessary, therefore inducing spin-up overheads frequently due to the constant load change, and resulting in increased execution time (132 min). It also introduces some overprovisioning, as it only requests the largest instances to constrain performance unpredictability. Because instances are released after remaining idle for a while, the average number of active instances is smaller than for SR. OdM does not overprovision allocations noticeably since it uses smaller instances, however, it significantly hurts performance, resulting in the scenario taking 48% more time. Performance degradation is partially the result of variability in the quality of instances, and of the high instance churn, due to their poor performance. 43% of obtained instances were released immediately after use.

The hybrid strategies (HF and HM) provision reserved resources for the minimum, steady-state load and use on-demand resources beyond that. HF needed in total 72 on-demand instances, of which 34 are used on average. HM needs a higher number of on-demand resources, because it uses smaller instances, and because it releases poorly-performing resources. 11% of on-demand instances obtained by HM were released immediately after use. Note that this is significantly lower than for OdM since only jobs that can tolerate performance unpredictability are assigned to on-demand resources. This issue is even less pronounced when

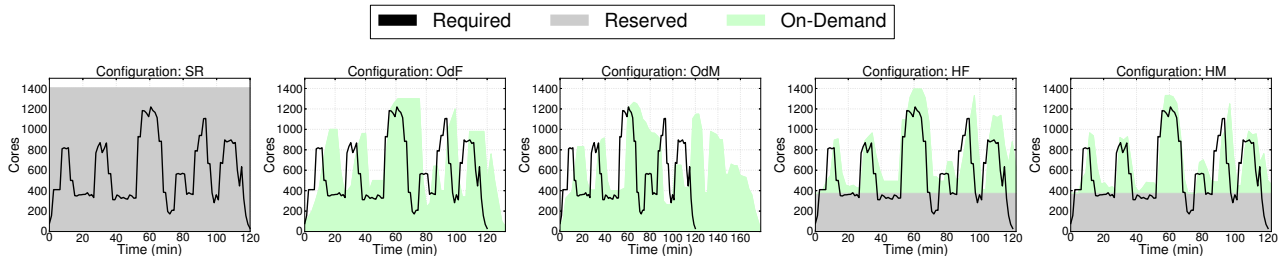


Figure 18: Resource allocation graphs for the five provisioning strategies in the case of the high variability scenario.

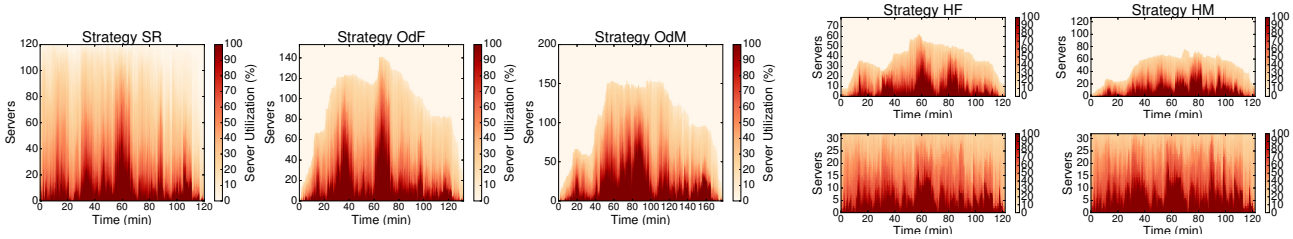


Figure 19: Resource utilization for the high variability scenario across the five provisioning strategies. Servers are ordered from most to least utilized at each point in time during the execution of the scenario.

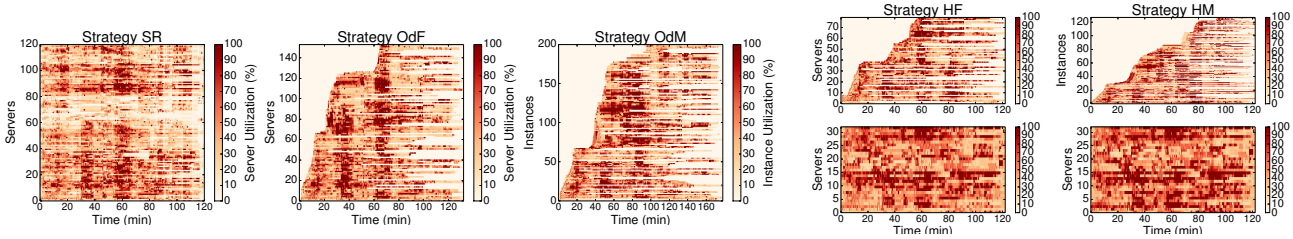


Figure 20: Resource utilization for the high variability scenario across the five provisioning strategies. Instances are ranked in the order in which they are obtained. For HF and HM we separate reserved (bottom) from on-demand (top) resources.

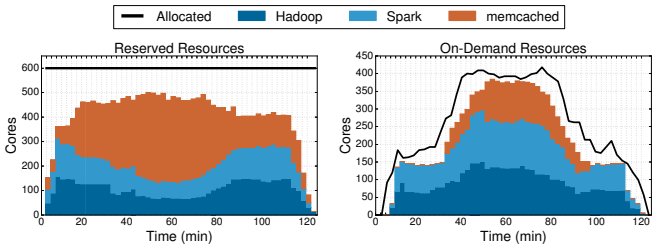


Figure 21: Breakdown of allocation per application type.

all on-demand resources are large instances (HF). Finally, spin-up overheads induce some delay in execution, which amounts to 2.6% over SR.

Figure 21 breaks down the allocation of the low variability scenario by application type, for HM. Initially the reserved resources are used for most jobs, until load reaches the soft utilization limit. Beyond that, the interference-sensitive memcached occupies most of the reserved resources, while batch workloads are mostly scheduled on the on-demand side. When the memcached load exceeds the capabilities of the reserved resources, part of its load is serviced by on-demand resources to avoid long queueing delays, although it is often allocated larger instances to meet its resource quality requirements.

5.5 Additional Provisioning Considerations

Spot instances: Spot instances consist of unallocated resources that cloud providers make available through a bidding interface. Spot instances do not have availability guarantees, and may be terminated at any point if the market price exceeds the bidding price for an instance type. Incorporating spot instances in provisioning for non-critical tasks or jobs with very relaxed performance requirements can further improve cost-efficiency. We will consider how spot instances interact with the current provisioning strategies in future work.

Reducing unpredictability: Resource partitioning (cache, memory or network bandwidth partitioning) can reduce unpredictability in fully on-demand systems. We plan to investigate how resource partitioning complements the current provisioning decisions.

Data management: In our current infrastructure both reserved and on-demand resources reside in the same physical cluster. When reserved resources are deployed as a private facility, provisioning must also consider how to minimize data transfers and replication across the two clusters.

6. Related Work

Cluster management: The prevalence of cloud computing has motivated several new designs for cluster management. Systems like Mesos [30], Torque [62], Tarcil [22], and Omega [57] all address the problem of allocating resources in large, shared clusters. Mesos is a two-level scheduler. It has a central coordinator that makes resource offers to application frameworks, and each framework has an individual scheduler that handles its assigned resources. Omega on the other hand, follows a shared-state approach, where multiple concurrent schedulers can view the whole cluster state, with conflicts being resolved through a transactional mechanism [57]. Tarcil leverages information on the type of resources applications need to employ a sampling-base distributed scheduler that returns high quality resources within a few milliseconds [22]. Dejavu identifies a few workload classes and reuses previous allocations for each class, to minimize reallocation overheads [63]. CloudScale [58], PRESS [26], AGILE [48] and the work by Gmach et al. [25] predict future resource needs online, often without a priori knowledge. Finally, auto-scaling systems, such as Rightscale [55], automatically scale the number of physical or virtual instances used by webserving workloads, to accommodate changes in user load.

A second line of work tries to identify the specific resources that are appropriate for incoming tasks [16, 18–20, 44, 46, 66]. Paragon uses classification to determine the impact of platform heterogeneity and workload interference on an unknown, incoming workload [17,18]. It then uses this information to achieve high application performance and high cluster utilization. Paragon, assumes that the cluster manager has full control over all resources, which is often not the case in public clouds. Nathuji et al. developed a feedback-based scheme that tunes resource assignments to mitigate memory interference [47]. Yang et al. developed an online scheme that detects memory pressure and finds colocations that avoid interference on latency-sensitive workloads [66]. Similarly, DeepDive detects and manages interference between co-scheduled workloads in a VM environment [49]. Finally, CPI2 [70] throttles low-priority workloads that induce interference to important, latency-critical services. In terms of managing platform heterogeneity, Nathuji et al. [46] and Mars et al. [43] quantified its impact on conventional benchmarks and Google services, and designed schemes to predict the most appropriate servers for a workload.

Hybrid clouds: Hybrid clouds consist of both privately-owned and publicly-rented machines and have gained increased attention over the past few years for several reasons, including cost-efficiency, future growth, as well as security and privacy concerns [2, 10, 31, 33, 69]. Breiter et al. [10] describe a framework that allows service integration in hybrid cloud environments, including actions such as overflowing in on-demand resources during periods of high load. Farahabady et al. [31] present a resource allocation strategy for hybrid clouds that attempts to predict the execution times of incoming jobs and based on these predictions generate Pareto-optimal resource allocations. Finally, Annapureddy et al. [2] and Zhang et al. [69] discuss the security challenges of hybrid environments, and propose ways to leverage the private portion of the infrastructure for privacy-critical computation.

Cloud economics: The resource pricing of cloud providers has been extensively analyzed. Ben-Yehuda et al. [9] contest whether the pricing strategy of spot instances on EC2 is indeed market-driven, and discuss alternative strategies. Deelman et al. [15] present cost-efficient cloud provisioning strategies for specific astronomy applications. Li et al. [38] compare the resource pricing of several cloud providers to assist users provision their applications. Finally, Guevara et al. [28] and Zahed et al. [68] incorporate the economics of heterogeneous resources in market-driven and game-theoretic strategies for resource allocation in shared environments.

7. Conclusions

We have discussed the resource offerings available on cloud providers today and showed their advantages and pitfalls with respect to cost, performance, and instantiation overheads. We then presented HCloud, a hybrid provisioning system that uses both reserved and on-demand resources. HCloud obtains information on the resource preferences of incoming jobs, and determines which jobs should be mapped to on-demand versus reserved resources and how many resources they should each receive. We showed that hybrid strategies can provide the best of both worlds in terms of performance and cost-efficiency; they preserve QoS for the majority of jobs, improve performance by 2.1x compared to fully on-demand resources, and reduce cost by 46% compared to fully reserved resources.

Acknowledgements

The authors sincerely thank Mendel Rosenblum, John Ousterhout, Daniel Sanchez, David Lo, and the anonymous reviewers for their feedback on earlier versions of this manuscript. This work was supported by the Platform Lab and NSF grant CNS-1422088. Christina Delimitrou was supported by a Facebook Graduate Fellowship.

References

[1] Amazon ec2. `http://aws.amazon.com/ec2/`.

[2] K. Annapureddy. Security challenges in hybrid cloud infrastructures. In *Aalto University, T-110.5290 Seminar on Network Security*. 2010.

[3] Autoscale. `https://cwiki.apache.org/cloudstack/autoscaling.html`.

[4] Aws autoscaling. `http://aws.amazon.com/autoscaling/`.

[5] G. Banga, P. Druschel, and J. Mogul. Resource containers: a new facility for resource management in server systems. In *Proceeedings of OSDI*. New Orleans, 1999.

[6] S. K. Barker and P. Shenoy. Empirical evaluation of latency-sensitive application performance in the cloud. In *Proceedings of the First Annual ACM SIGMM Conference on Multimedia Systems (MMsys)*. Scottsdale, AR, 2010.

[7] L. Barroso. Warehouse-scale computing: Entering the teenage decade. *ISCA Keynote, SJ, June 2011*.

[8] L. Barroso and U. Hoelzle. *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*. MC Publishers, 2009.

[9] O. A. Ben-Yehuda, M. Ben-Yehuda, A. Schuster, and D. Tsafrir. Deconstructing amazon ec2 spot instance pricing. In *ACM TEAC, 1(3), September 2013*.

[10] G. Breiter and V. Naik. A framework for controlling and managing hybrid cloud service integration. In *Proceedings of the 2013 IEEE International Conference on Cloud Engineering (IC2E)*. Redwood City, CA, 2013.

[11] M. Carvalho, W. Cirne, F. Brasileiro, and J. Wilkes. Long-term slos for reclaimed cloud computing resources. In *Proceedings of ACM Symposium on Cloud Computing (SOCC)*. Seattle, WA, 2014.

[12] J. Chase, D. Anderson, P. Thakar, A. Vahdat, and R. Doyle. Managing energy and server resources in hosting centers. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*. Banff, CA, 2001.

[13] Linux containers. `http://lxc.sourceforge.net/`.

[14] J. Dean and L. A. Barroso. The tail at scale. In *Communications of the ACM (CACM), Vol. 56 No. 2, Pages 74-80*.

[15] E. Deelman, G. Singh, M. Livny, B. Berriman, and J. Good. The cost of doing science on the cloud: The montage example. In *Proceedings of Supercomputing (SC)*. Austin, TX, 2008.

[16] C. Delimitrou, N. Bambos, and C. Kozyrakis. QoS-Aware Admission Control in Heterogeneous Datacenters. In *Proceedings of the International Conference of Autonomic Computing (ICAC)*. San Jose, CA, USA, 2013.

[17] C. Delimitrou and C. Kozyrakis. iBench: Quantifying Interference for Datacenter Workloads. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*. Portland, OR, September 2013.

[18] C. Delimitrou and C. Kozyrakis. Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Houston, TX, 2013.

[19] C. Delimitrou and C. Kozyrakis. QoS-Aware Scheduling in Heterogeneous Datacenters with Paragon. In *ACM Transactions on Computer Systems (TOCS), Vol. 31 Issue 4*. December 2013.

[20] C. Delimitrou and C. Kozyrakis. Quality-of-Service-Aware Scheduling in Heterogeneous Datacenters with Paragon. In *IEEE Micro Special Issue on Top Picks from the Computer Architecture Conferences*. May/June 2014.

[21] C. Delimitrou and C. Kozyrakis. Quasar: Resource-Efficient and QoS-Aware Cluster Management. In *Proceedings of the Nineteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Salt Lake City, UT, USA, 2014.

[22] C. Delimitrou, D. Sanchez, and C. Kozyrakis. Tarcil: Reconciling Scheduling Speed and Quality in Large Shared Clusters. In *Proceedings of the Sixth ACM Symposium on Cloud Computing (SOCC)*. Kohala Coast, HI, 2015.

[23] B. Farley, A. Juels, V. Varadarajan, T. Ristenpart, K. D. Bowers, and M. M. Swift. More for your money: Exploiting performance heterogeneity in public clouds. In *Proceedings of the ACM Symposium on Cloud Computing (SOCC)*. San Jose, CA, 2012.

[24] Google compute engine. `https://developers.google.com/compute/`.

[25] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Workload analysis and demand prediction of enterprise data center applications. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC)*. Boston, MA, 2007.

[26] Z. Gong, X. Gu, and J. Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *Proceedings of the International Conference on Network and Service Management (CNSM)*. Niagara Falls, ON, 2010.

[27] G. R. Grimmett and D. R. Stirzaker. Probability and random processes. 2nd Edition. Clarendon Press, Oxford, 1992.

[28] M. Guevara, B. Lubin, and B. Lee. Navigating heterogeneous processors with market mechanisms. In *Proceedings of the IEEE Symposium on High Performance Computer Architecture (HPCA)*. Shenzhen, China, 2013.

[29] J. Hamilton. Cost of power in large-scale data centers. `http://perspectives.mvdirona.com`.

[30] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. Joseph, R. Katz, S. Shenker, and I. Stoica. Mesos: A platform for fine-grained resource sharing in the data center. In *Proceedings of the USENIX Sumposium on Networked Systems Design and Implementation (NSDI)*. Boston, MA, 2011.

[31] M. Hoseinyfarahabady, H. Samani, L. Leslie, Y. C. Lee, and A. Zomaya. Handling uncertainty: Pareto-efficient bot scheduling on hybrid clouds. In *Proceedings of the International Conference for Parallel Processing (ICPP)*. Lyon, France, 2013.

[32] A. Iosup, N. Yigitbasi, and D. Epema. On the performance

variability of production cloud services. In *Proceedings of IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. Newport Beach, CA, 2011.

[33] V. Khadilkar, K. Y. Oktay, B. Hore, M. Kantarcioglu, S. Mehrotra, and B. Thuraisingham. Risk-aware data processing in hybrid clouds. TR-UTDCS-31-11, 2011.

[34] Y. E. Khamra, H. Kim, S. Jha, and M. Parashar. Exploring the performance fluctuations of hpc workloads on clouds. In *Proceeedings of IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*. Indianapolis, IN, 2010.

[35] L. Kleinrock. Queueing systems volume 1: Theory. pp. 101-103, 404.

[36] E. Lau, J. E. Miller, I. Choi, D. Yeung, S. Amarasinghe, and A. Agarwal. Multicore performance optimization using partner cores. In *Proceedings of the USENIX Workshop on Hot Topics in Parallelism (HotPar)*. Berkeley, CA, 2011.

[37] J. Laudon. Performance/watt: the new server focus. In *ACM SIGARCH Computer Architecture News: dasCMP*. Vol. 33 Issue 4, p. 5-13, November 2005.

[38] A. Li, X. Yang, S. Kandula, and M. Zhang. Cloudcmp: Comparing public cloud providers. In *Proceeedings of Internet Measurement Conference (IMC)*. Melbourne, Australia, 2010.

[39] Host server cpu utilization in amazon ec2 cloud. `http://goo.gl/nCfYFX`.

[40] J. Lorch and A. Smith. Reducing processor power consumption by improving processor time management in a single-user operating system. In *Proceeedings of Annual International Conference on Mobile Computing and Networking (Mobicom)*. New York, NY, 1996.

[41] Mahout. `http://mahout.apache.org/`.

[42] D. Mangot. Ec2 variability: The numbers revealed. `http://goo.gl/NAH2lm`.

[43] J. Mars and L. Tang. Whare-map: heterogeneity in "homogeneous" warehouse-scale computers. In *Proceeedings of the International Symposium on Computer Architecture (ISCA)*. Tel-Aviv, Israel, 2013.

[44] J. Mars, L. Tang, R. Hundt, K. Skadron, and M. L. Soffa. Bubble-up: increasing utilization in modern warehouse scale computers via sensible co-locations. In *Proceeedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO)*. Porto Alegre, Brazil, 2011.

[45] S. M. Martin, K. Flautner, T. Mudge, and D. Blaauw. Combined dynamic voltage scaling and adaptive body biasing for lower power microprocessors under dynamic workloads. In *Proceeedings of International Conference On Computer Aided Design (ICCAD)*. 2002.

[46] R. Nathuji, C. Isci, and E. Gorbatov. Exploiting platform heterogeneity for power efficient data centers. In *Proceeedings of the International Conference on Autonomic Computing (ICAC)*. Jacksonville, FL, 2007.

[47] R. Nathuji, A. Kansal, and A. Ghaffarkhah. Q-clouds: Managing performance interference effects for qos-aware clouds. In *Proceeedings of EuroSys France*, 2010.

[48] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes. Agile: Elastic distributed resource scaling for infrastructure-as-a-service. In *Proceedings of the International Conference on Autonomic Computing (ICAC)*. 2013.

[49] D. Novakovic, N. Vasic, S. Novakovic, D. Kostic, and R. Bianchini. Deepdive: Transparently identifying and managing performance interference in virtualized environments. In *Proceeedings of the USENIX Annual Technical Conference (ATC)*. San Jose, CA, 2013.

[50] S. Ostermann, A. Iosup, N. Yigitbasi, R. Prodan, T. Fahringer, and D. Epema. A performance analysis of ec2 cloud computing services for scientific computing. In *Lecture Notes on Cloud Computing*. Volume 34, p.115-131, 2010.

[51] Z. Ou, H. Zhuang, J. K. Nurminen, A. Ylä-Jääski, and P. Hui. Exploiting hardware heterogeneity within the same instance type of amazon ec2. In *HotCloud*. 2012.

[52] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. Sparrow: Distributed, low latency scheduling. In *Proceeedings of the Symposium on Operating Systems Principles (SOSP)*. Farminton, PA, 2013.

[53] M. Rehman and M. Sakr. Initial findings for provisioning variation in cloud computing. In *Proceeedings of CloudCom*. Indianapolis, IN, 2010.

[54] C. Reiss, A. Tumanov, G. Ganger, R. Katz, and M. Kozych. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceeedings of ACM Symposium on Cloud Computing (SOCC)*. 2012.

[55] Rightscale. `https://aws.amazon.com/solution-providers/isv/rightscale`.

[56] J. Schad, J. Dittrich, and J.-A. Quiané-Ruiz. Runtime measurements in the cloud: Observing, analyzing, and reducing variance. *Proceeedings VLDB Endow.*, 3(1-2):460–471, Sept. 2010.

[57] M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes. Omega: flexible, scalable schedulers for large compute clusters. In *Proceeedings of EuroSys*. 2013.

[58] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceeedings of ACM Symposium on Cloud Computing (SOCC)*. Cascais, Portugal, 2011.

[59] T. Simunic and S. Boyd. Managing power consumption in networks on chips. In *Proceeedings of Design Automation Conference (DAC)*. Paris, France, 2002.

[60] S. Swanson, A. Putnam, M. Mercaldi, K. Michelson, A. Petersen, A. Schwerin, M. Oskin, and S. J. Eggers. Area-performance trade-offs in tiled dataflow architectures. In *Proceeedings of ACM SIGARCH Computer Architecture News, v.34 n.2, p.314-326, May 2006*.

[61] K. Therdsteerasukdi, G. Byun, J. Cong, F. Chang, and G. Reinman. Utilizing rf-i and intelligent scheduling for better throughput/watt in a mobile gpu memory system. In *Transactions on Architecture and Code Optimization (TACO) 8(4)*. 2012.

[62] Torque resource manager. http://www.adaptivecomputing.com/products/open-source/torque/.

[63] N. Vasić, D. Novaković, S. Miučin, D. Kostić, and R. Bianchini. Dejavu: accelerating resource allocation in virtualized environments. In *Proceedings of the Seventeenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. 2012.

[64] G. Wang and T. S. E. Ng. The impact of virtualization on network performance of amazon ec2 data center. In *Proceeedings of IEEE International Conference on Computer Communications (INFOCOM)*. San Diego, CA, 2010.

[65] Windows azure. http://www.windowsazure.com/.

[66] H. Yang, A. Breslow, J. Mars, and L. Tang. Bubble-flux: precise online qos management for increased utilization in warehouse scale computers. In *Proceeedings of the International Symposium on Computer Architecture (ISCA)*. 2013.

[67] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. Franklin, S. Shenker, and I. Stoica. Spark: Cluster computing with working sets. In *Proceedings of the USENIX Sumposium on Networked Systems Design and Implementation (NSDI)*. San Jose, CA, 2012.

[68] S. M. Zahed and B. C. Lee. Ref: Resource elasticity fairness with sharing incentives for multiprocessors. In *Proceeedings of the Nineteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*. Salt Lake City, UT, 2014.

[69] J. Y. Zhang, P. Wu, J. Zhu, H. Hu, and F. Bonomi. Privacy-preserved mobile sensing through hybrid cloud trust framework. In *Proceeedings of IEEE International Conference on Cloud Computing (CLOUD)*. 2013.

[70] X. Zhang, E. Tune, R. Hagmann, R. Jnagal, V. Gokhale, and J. Wilkes. Cpi2: Cpu performance isolation for shared compute clusters. In *Proceeedings of EuroSys*. Prague, 2013.

[71] L. Zhao, R. Iyer, J. Moses, R. Illikkal, S. Makineni, and D. Newell. Exploring large-scale cmp architectures using manysim. In *IEEE Micro, vol.27 n.4, July 2007*.