STANFORD UNIVERSITY

COMPUTER SCIENCE TECHNICAL REPORT

CSTR 2014-06 11/25/2014

Optimizing Resource Provisioning in Shared Cloud Systems

Christina DELIMITROU

Christos KOZYRAKIS

November 25, 2014

Stanford University

Optimizing Resource Provisioning in Shared Cloud Systems

Abstract

Cloud computing promises flexibility and high performance for users and cost efficiency for operators. To achieve this premise, cloud providers offer several provisioning strategies including long-term reserved resources and short-term on-demand resources. Determining the most appropriate provisioning strategy is a complex, multi-dimensional problem that depends on the load fluctuation, interference sensitivity and duration of incoming jobs and the performance unpredictability and cost of the provisioned 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 mixes of batch and latency-critical applications and increasing levels of load variability. We show that either approach is suboptimal from the performance or cost perspective. We then explore hybrid provisioning strategies with both reserved and on-demand resources. We design policies that account for the resource preferences of incoming jobs to automatically determine which jobs should be mapped to reserved versus on-demand resources based on overall load, and resource unpredictability. We demonstrate that hybrid configurations improve both performance and cost-efficiency compared to fully reserved and fully on-demand systems. Specifically they 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.

1. Introduction

An increasing amount of computing is now hosted in public clouds, such as Amazon's EC2 [1], Windows Azure [63] and Google Compute Engine [20], or in private clouds managed by frameworks such as VMware vCloud [61], OpenStack [46], and Mesos [26]. Cloud platforms provide two major advantages for end-users and cloud operators: *flexibility* and *cost efficiency* [7, 8, 25]. 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 can provision resources for their applications in two basic manners; using *reserved* and *on-demand* resources. *Reserved resources* consist of servers reserved for long periods of time (typically 1-3 years [1]) and offer consistent service, but come at a significant upfront cost for the purchase of the long-term resource contract. In the other extreme are *on-demand resources*, which can be full servers or smaller instances and are progressively obtained as they become necessary. In this case, the user pays only for resources used at each point in time, but the per hour cost is 2-3x higher compared to reserved resources. Moreover, acquiring on-demand resources induces instantiation overheads and depending on the type of instance, the variability in the quality of service obtained can be significant.

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, 19, 28, 30, 34, 38, 47, 48, 50, 53, 62]. Figure 1 shows the variability in performance for a Hadoop job running a recommender system using Mahout [37] on various instance types on Amazon EC2 [1] and on Google Compute Engine (GCE) [20]. Analytics such as Hadoop and Spark [65] 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 instances (`mX`) [1, 20]. 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

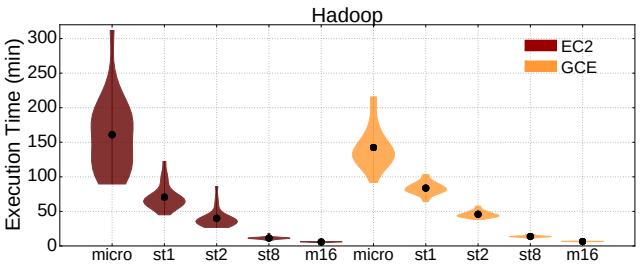


Figure 1: Performance unpredictability on Amazon EC2 and Google Compute Engine for a Hadoop job.

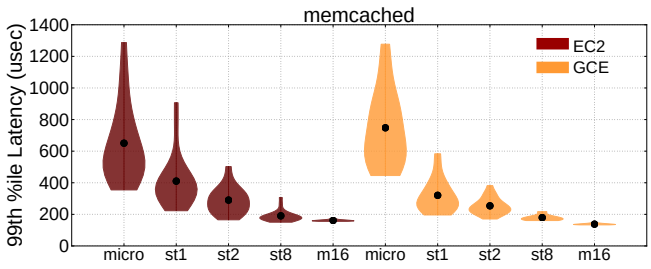


Figure 2: Performance unpredictability on Amazon EC2 and Google Compute Engine for memcached.

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.

and resource management optimized performance per Watt for small- and large-scale systems [32, 33, 57, 58, 69]. 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) [20] using three representative workload scenarios with mixes of 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 for each scenario, 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 beneficial 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, especially when using smaller instances. They also incur instantiation overheads to spin-up new VMs. Our study also shows that to achieve reasonable performance predictability with either strategy, it is crucial to understand the resource preferences and sensitivity to interference of individual applications [18, 39, 51]. 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 each job's performance constraint [18].

Next, we consider *hybrid provisioning strategies* that use both reserved (long-term) and on-demand (short-term) resources. A hybrid provisioning strategy has the potential to offer the best of both worlds by allowing users to leverage reserved resources for the steady-state long-term load, and on-demand resources for short-term resource needs. The main challenge with hybrid provisioning strategies is determining how to schedule jobs between the two types of resources. We show that leveraging the knowledge on resource preferences and accounting for the characteristics of on-demand resources, and the system load enables correct mapping of jobs to reserved and on-demand resources. Table 1 shows the differences between the three main provisioning strategies with respect to *cost*, *performance unpredictability*, *instantiation overheads* and *provisioning flexibility*.

We demonstrate that hybrid provisioning strategies achieve both high 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 the fully reserved and fully on-demand strategies examined before over a wide spectrum of workload scenarios. Hybrid provisioning 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 during steady-state. Finally, we perform a detailed sensitivity analysis of performance and cost with job parameters, such as duration, and system parameters such as resource pricing, spin-up overhead, and external load.

2. Cloud Workloads and Systems

2.1. Workload Scenarios

We examine the three workload scenarios shown in Figure 3 and summarized in Table 2. Each scenario consists of a mix of batch applications (Hadoop workloads running over Mahout [37] and Spark jobs) and latency-critical workloads (memcached). The batch jobs are machine learning and data mining applications, including recommender systems, support vector

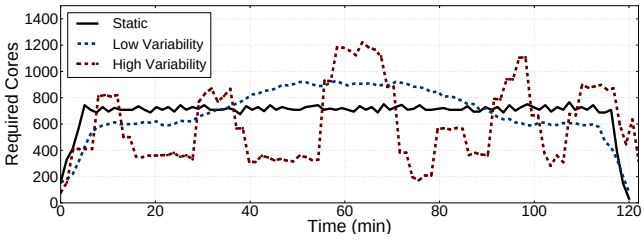


Figure 3: The three workload scenarios.

	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.

machines, matrix factorization, and linear regression. 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 size of the dataset.

The first scenario has minimal load variability (*Static*). In steady-state the aggregate resource requirements are 854 cores on average. Approximately 55% of cores are required for batch jobs and the remaining 45% for the 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 job is shorter (8.1 min duration on average).

The ideal duration for each scenario, with no scheduling delays or degradation due to interference between workloads, is approximately 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 external interference by imposing external load that fluctuates $\pm$ 10% around a 25% average utilization [8, 18]. The external load is generated using both batch and latency-critical workloads. Section 5.1 includes a sensitivity study to the intensity of 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 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 [20]. Although sustained usage 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 and adapt it to the resource prices of GCE. Specifically, we approximate the cost of reserved resources on GCE based on the reserved to on-demand price ratio for EC2, adjusted to the instance 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 types of resource offerings in cloud systems are *reserved* and *on-demand* resources. Reserved instances require a high upfront capital investment, but have 2-3x lower per-hour cost than on-demand resources, offer better service availability (1-year minimum), and provide consistent performance. On-demand resources are charged in a pay-as-you-go manner, but incur spin-up overheads and experience performance unpredictability due to interference from external load. We ignore spot instances for the purpose of this work, since they do not provide any availability guarantees.

The provisioning strategy must acquire the right type and number of resources for a workload scenario. 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 obvious provisioning strate-

gies described in Table 3: a statically-provisioned strategy using only reserved resources (SR); an on-demand strategy (OdF) that only uses full servers (16 vCPU instances); and an on-demand strategy (OdM) that uses instances of any size and type.

3.1. Statically Reserved Resources (SR)

This strategy statically provisions reserved resources for a 1 year period, the shortest contract for reserved resources on cloud systems such as EC2. Reserved resources require significant capital investment upfront, although the *per-hour* charge is 2-3x lower than for the corresponding on-demand instances. Moreover, reserved resources are readily available as jobs arrive, eliminating the overhead of spinning up new VMs on-demand. Because SR only reserves large instances (16 vCPU), 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 workload 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 [7,8,14,31]. We explain the insight behind the amount of overprovisioning in Section 3.3. Peak requirements can be easily estimated for mostly static workload scenarios. For scenarios with load variability, static provisioning results in acquiring a large number of resources which remain underutilized for significant periods of time.

3.2. Dynamic On-Demand Resources (OdF, OdM)

We now examine two provisioning strategies that acquire resources as they become necessary to accommodate the incoming jobs of each workload scenario. In this case there is no need for a large expenditure upfront, but the price of each instance per hour is 2-3x higher compared to the corresponding reserved resources. Moreover, each new instance now incurs the overhead needed to spin up the new VMs. This is typically 12-19 seconds for GCE, although the 95^{th} percentile of the spin-up overhead is up to 2 minutes. Smaller instances tend to incur higher spin-up overheads.

Because of spin-up overheads, these two strategies must also decide how long they should retain the resources for after a job completes. If, for example, a workload scenario has no or little load variability, instances should be retained to amortize the spin-up overhead. On the other hand, retaining instances when load variability is high can result in underutilized resources. We determine retention time by drawing from related work on processor power management. The challenge in that case is to determine when to switch to low power modes that enable power savings but incur overheads to revert to an active mode [36,41,56]. Given that the job inter-arrival time in our scenarios is 1 second, we set the retention time to 10x the spin-up overhead of an instance. [1] Section 5.1 shows a sensitivity analysis to retention time. Only instances that perform in a satisfactory manner are retained past the completion of their jobs.

	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.

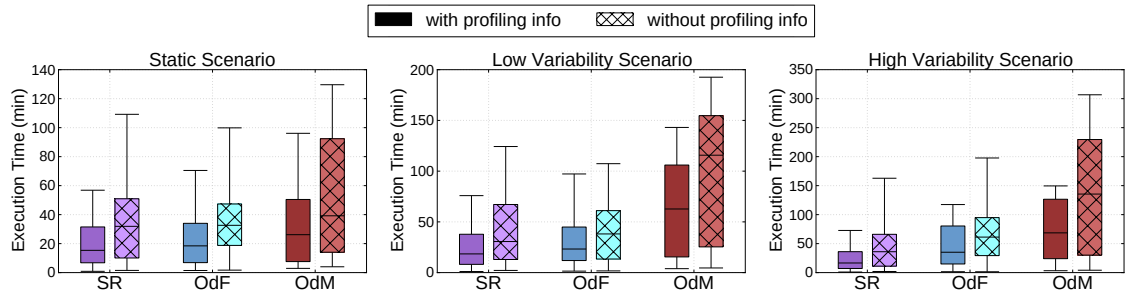
We examine two variants of on-demand provisioning strategies. On-demand Full (OdF) only uses large instances (16 vCPUs), which are much less prone to external interference (see Section 1). On-demand Mixed (OdM) acquires on-demand resources of any instance type, including smaller instances with 1-8 vCPUs. While OdM offers more flexibility, it introduces the issue that performance unpredictability due to external interference now becomes substantial. There are ways to improve performance predictability in fully on-demand provisioning strategies, e.g., by sampling multiple instances for each required instance and only keeping the better-behaved instances [19]. Although this approach addresses the 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.

3.3. The Importance of Resource Preferences

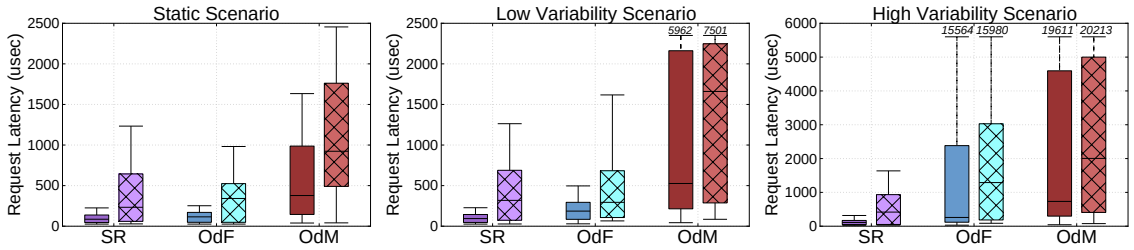
So far, we have assumed that the provisioning strategy has limited knowledge about the resource preferences of individual jobs within a workload scenario. Traditionally, the end-users have to specify how many resources each job should use; unfortunately this is known to be error-prone and to frequently lead to significant resource overprovisioning [8,11,18,35,51]. Moreover, this 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 the statically-reserved and on-demand strategies, which will acquire more/less resources than what is truly needed by an application. The lack of interference understanding is equally problematic. SR will likely colocate jobs that interfere negatively with each other on the same instance. OdF and OdM will likely acquire instance types that are prone to higher interference than what certain jobs can tolerate.

The recently-proposed Quasar system provides a methodology to quickly determine the resource preferences of new jobs [18]. When a job is submitted to the system, it is first profiled on two instance types, while injecting interference in two shared resources, e.g., last level cache and network bandwidth.

[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 workload scenarios with the three provisioning 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 shows the mean.

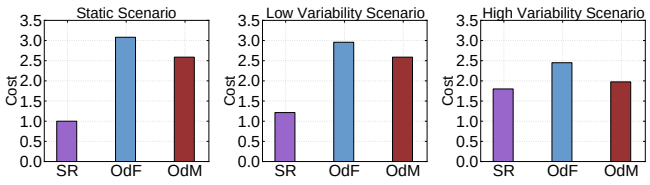


Figure 5: Cost of fully reserved and on-demand systems.

This profiling 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 [16, 18]. Large values of c_i 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 an order preserving encoding scheme as follows: $Q = c_j \cdot 10^{(2 \cdot (N-1))} + c_k \cdot 10^{(2 \cdot (N-2))} + ... + c_n$, 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 interference in shared resources.

We use Quasar's estimations of resource preferences and interference sensitivity to improve resource provisioning. For SR, we use these estimations to find the most suitable resources available in the reserved instances with respect to resource size and interference using a simple greedy search [18]. Accounting for the information on 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 in this case.

Finally, we must detect suboptimal application performance and revisit the allocation decisions at runtime [3, 4, 12, 18, 52]. Once an application is scheduled its performance is monitored and compared against its expected QoS. If performance drops below QoS we take action [18]. At a high level, we first try to restore performance through local actions, e.g., increasing the resource allocation, and then through rescheduling. Rescheduling is very 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 individual job preferences. Figure 4 shows the performance achieved by each of the three provisioning 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 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 percentile and the horizontal line shows the mean. When the information from Quasar is not used, the re-

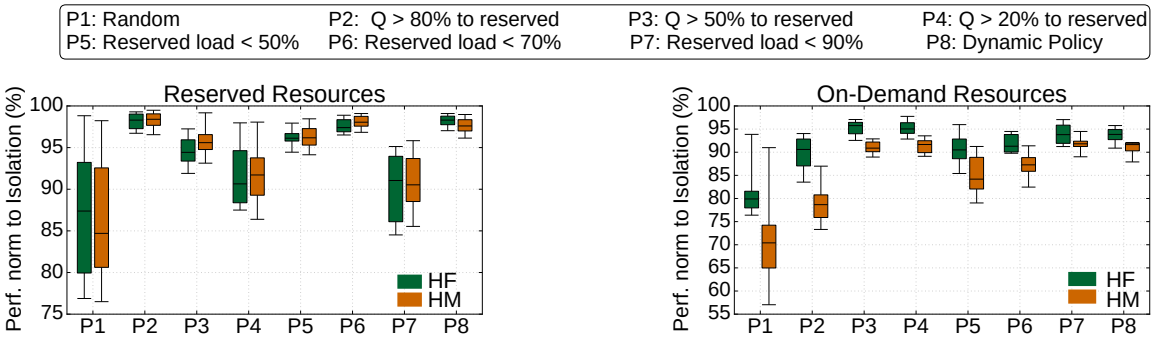


Figure 6: Sensitivity to the policy of mapping jobs to reserved versus on-demand resources for HF and HM.

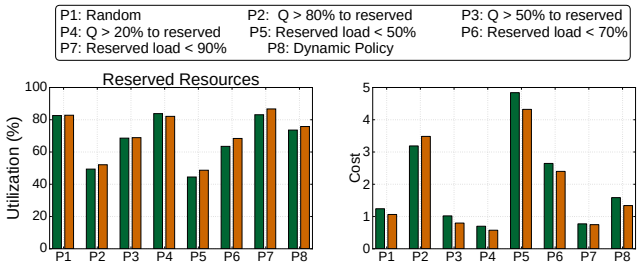


Figure 7: Utilization of reserved resources and cost with different application mapping policies for HF and HM.

sources for each job are sized based on user-defined resource 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 load [51]. OdM requests the smallest instance size that satisfies the resource demands of a job. SR allocates resources for workloads on the reserved instances with the most available resources (least-loaded).

It is clear from Figure 4 that all three provisioning 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 in the case of latency-critical applications, where the performance metric of interest is tail, instead of average performance. Omitting the information on interference sensitivity in this case significantly hurts request latency. In all following results, we assume that provisioning takes job preferences into account, unless otherwise stated.

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 workloads. OdF behaves near-optimally for the static scenario, but worsens for the scenarios where variability is present. The main reason is the spin-up overhead required to obtain new resources as they become necessary. Strategy OdM achieves the worst performance of all three provisioning strategies for every scenario (2.2x worse than SR on average), in part because of the spin-up overhead, 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 particularly cost effective in the presence of high workload variability, since it results in significant over-provisioning. Between the two on-demand strategies, OdM incurs lower cost, since it uses smaller instances, while OdF only uses the largest instances available. Note however that the cost savings of OdM translate to a significant performance degradation due to resource 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 resourc-es have the potential to achieve the best of both worlds. This section presents two hybrid provisioning strategies that intelligently assign jobs between reserved and on-demand resources and compares their performance and cost against the strategies of Section 3. Again, we make use of the information on resource preferences and interference sensitivity of individual jobs, as estimated by Quasar.

4.1. Provisioning Strategies

We design two hybrid strategies that use both reserved and on-demand resources. The first strategy (HF) only uses large instances for the on-demand resources, to reduce performance 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 of on-demand resources is the same as for the purely on-demand strategies OdF and OdM. The reserved resources in both cases are large instances, as with the

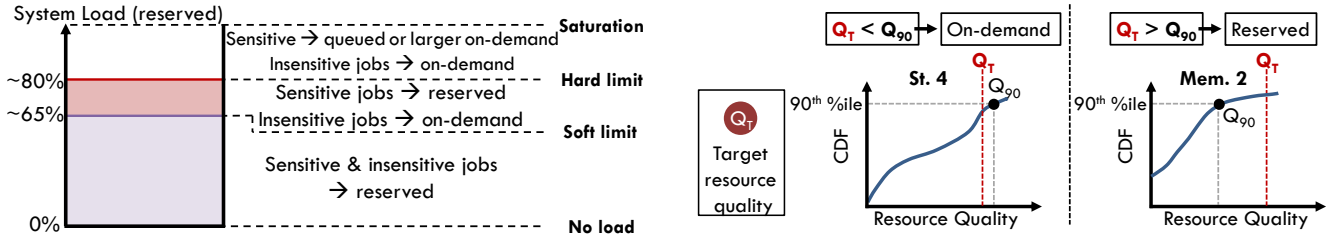


Figure 8: Application mapping scheme between reserved and on-demand instances for HF and HM.

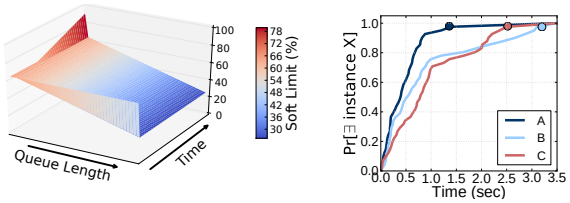


Figure 9: Determining the soft utilization limit (left) and the expected waiting time (right) in HF and HM.

statically-provisioned strategy (SR). We configure the number of reserved instances to accommodate the minimum steady-state load, e.g., 600 cores for the low variability scenario to avoid overprovisioning of reserved resources. For scenarios with low steady-state load but high load variability the majority of resources will be on-demand.

Since HF uses large instances with limited performance unpredictability for both reserved and on-demand resources, it mostly uses on-demand instances to serve overflow load. In contrast, with HM on-demand instances may be smaller and can experience resource interference from external load. Therefore, for hybrid strategies it is critical to determine which jobs should be mapped to reserved versus on-demand resources, based on their interference sensitivity and the availability of reserved resources.

4.2. Application Mapping Policies

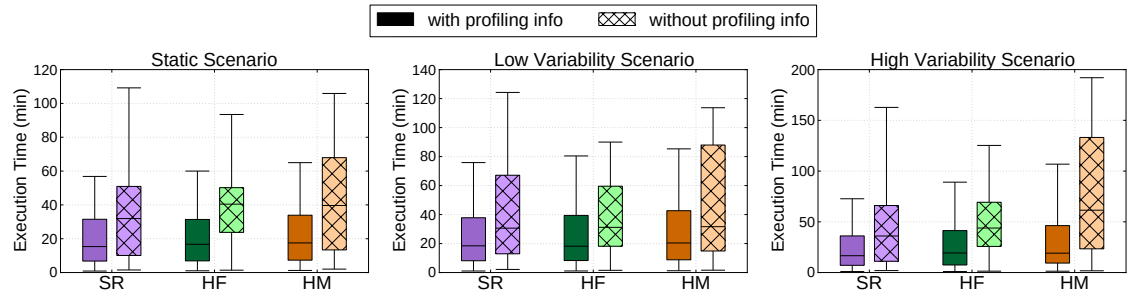
We first consider a baseline policy that maps applications between the reserved and on-demand resources randomly using a fair coin. Figure 6 shows the performance of applications mapped to the reserved (left) and on-demand resources (right) for the two hybrid provisioning strategies in the case of the high variability scenario. Performance is normalized to the performance each job achieves if it runs with unlimited resources alone in the system (in isolation). Figure 7 also shows the utilization of the reserved instances and the total cost to run the 2 hour scenario normalized to the cost of the static scenario with SR. Because of the large number of scheduled applications, approximately half of them will be scheduled on reserved and half on on-demand resources [23]. The random policy hurts performance for jobs mapped to either type of resources. In the reserved resources, performance degrades as more workloads than the instances can accommodate are assigned to them, and are therefore queued. In the on-demand resources, performance degrades for two reasons. First, because of the inherent unpredictability of resources, especially in the case of HM, and, more prominently, because jobs that are sensitive to interference and should have been mapped to reserved resources slow down 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 the on-demand resources. As we move from $P2$ to $P4$ the performance of jobs in the on-demand instances improves, as the number of applications mapped to them decreases. In contrast, the performance of jobs scheduled to reserved resources worsens due to increased demand and queueing for resources. In general, performance is worse for HM in the on-demand resources, due to the increased performance variability of smaller instances.

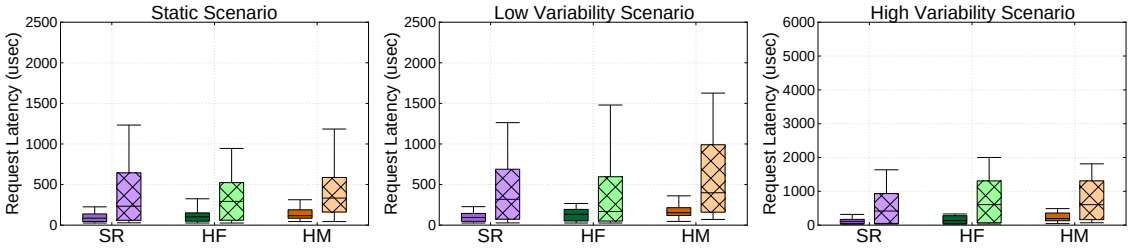
It is clear that there needs to be an upper load limit for the reserved resources. The next three policies $P5 - P7$ set progressively higher, static limits. For low utilization limits, e.g., 50-70% the performance of jobs on reserved resources is near-optimal. In contrast, jobs assigned to on-demand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

Figure 8 explains the dynamic policy. We set two utilization



(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 provisioning 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 shows the mean.

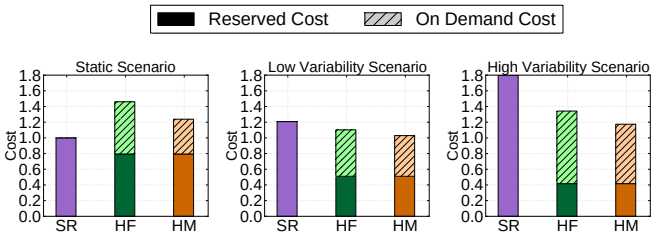


Figure 11: Cost comparison between SR, HF and HM.

limits for the 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 applications 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.

Second, we set a *hard limit* for utilization, when jobs need to get queued before reserved resources become available. At this point, any jobs for which on-demand resources are satisfactory are scheduled in the on-demand instances and all remaining jobs are locally queued [49]. An exception occurs for jobs whose queueing time is expected to exceed the time it would take to spin up 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 applications get queued. If the number of queued jobs increases 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 incoming jobs. We use a simple feedback loop with linear transfer functions to adjust the soft utilization limit of the reserved instances as a workload scenario progresses. Figure 9a shows how the soft limit changes with execution time and queue length.

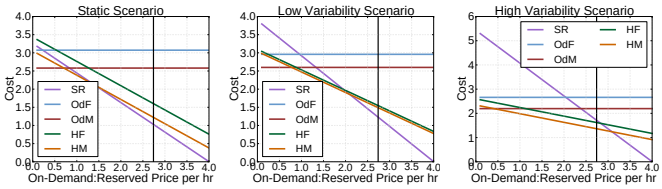


Figure 12: Sensitivity to on-demand:reserved cost.

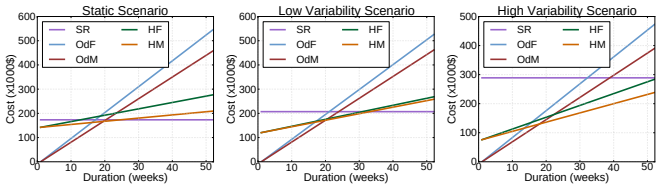


Figure 13: Sensitivity to scenario duration.

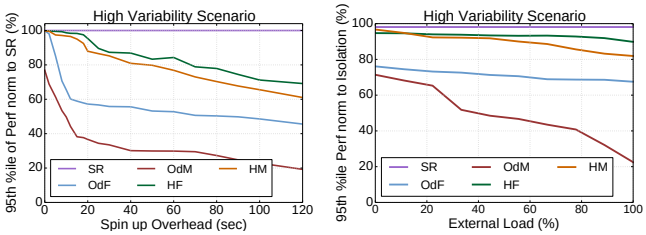


Figure 14: Sensitivity to spin-up time and external load.

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 the profiling information for new jobs. Again we separate batch from latency-critical jobs. As expected, having 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 the 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 the scenarios with significant load variability. The main reason why HF and HM achieve good performance is that they differentiate between applications 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 provisioning 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, statically-provisioned resources is not cost-efficient both due to long-term reservations, 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 achieves higher cost-efficiency since it uses smaller instances.

5. Discussion

5.1. Sensitivity to Job/System Parameters

We first evaluate the sensitivity of the previous findings to various system and workload parameters. Unless otherwise specified, we use the same strategies as before to provision reserved and/or on-demand resources.

Resource cost: The current average cost ratio of on-demand to reserved resource per hour is 2.74. Figure 12 shows how the relative cost of the three scenarios varies for each of the five strategies when this ratio changes. The current ratio is shown with a vertical line at 2.74. All costs are normalized to the cost for the static scenario using SR. We change the ratio by scaling the price of reserved resources. We vary the ratio in $[0.01, 4]$; beyond that point the cost of SR per hour becomes practically negligible. Initially (0.01), strategies using only on-demand resources (OdF, OdM) are significantly more cost-efficient, especially for the scenarios with load 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 needs to become 3 for SR to be more cost-efficient per hour than HM). Note that SR still requires at least a 1-year commitment, in contrast to the on-demand strategies. Finally, the hybrid strategies achieve the lowest per-hour cost for significant ranges of the price ratio, especially for scenarios with load variability.

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

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

External load: Figure 14b shows the sensitivity of performance to external load (load in machines due to jobs beyond those provisioned with our strategies). SR provisions a fully-controlled environment, therefore there is no external load to affect performance. OdF and HF are also tolerant to external load, as they only use the largest instances, which are much less prone to external interference. For HM performance degrades minimally until 50% load, beyond which point the estimations on resource quality become inaccurate. OdM suffers most of the performance degradation as all of its resources are susceptible to external interference.

Retention time: Figure 15 shows the 95^{th} percentile of performance and the cost of each strategy, as the time for which idle instances are maintained changes for the high variability scenario. As expected, releasing well-behaved instances immediately hurts performance, as it increases the overheads from spinning-up new resources. This is especially the case for this scenario, where load changes frequently. With respect to cost, higher retention time increases the cost of strategies using only on-demand resources (OdF, OdM), while SR remains unchanged; the difference for hybrid strategies is small. An unexpected finding is that excessive resource retention slightly hurts performance for OdM and HM. The primary reason is the temporal variability in the quality of on-demand resources, which degraded by the time new applications were assigned to these instances.

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 the classification engine takes 5-10 sec, but only needs to happen the first time a job is submitted. Classification itself takes 50msec on average. Decision overheads include the greedy scheduler in the statically-reserved strategy (SR) and the overhead of deciding whether to schedule a new job on reserved versus 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

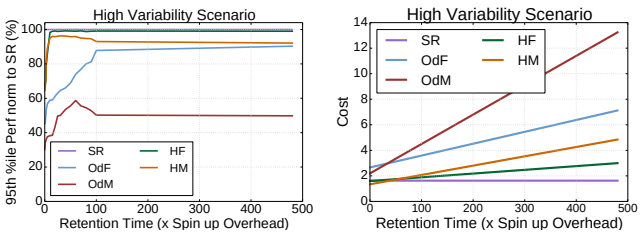


Figure 15: Sensitivity to resource retention time.

to the execution time of jobs on average.

5.3. Different Pricing Models

So far we have assumed a pricing model for reserved and on-demand instances similar to the one used by 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 of a type for a fraction of the month, the lower the per-hour instance price for the remainder of the month. This approach does not differentiate whether one uses a single instance of type A for 3 weeks or 3 instances of type A for 1 week each. Microsoft Azure only offers on-demand resources of different types.

Even without reserved resources, the problem of selecting the appropriate instance size and configuration, and determining how long to keep an instance for before releasing it remains. Figure 16 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 a one month period, so that GCE discounts can take effect. Cost is normalized to the cost of the static workload scenario under the SR provisioning strategy using the *reserved & on-demand* pricing model. Even with these alternative pricing models using the hybrid strategies and accounting for the resource preferences of incoming applications to optimize provisioning significantly benefits cost. For example, for the high variability scenario HM achieves 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 level of usage from the specific instance used. For example, monthly usage is considered the same between a single instance used for 50% of the month, and N instances of the same type used for $(50/N)\%$ of the month each. This introduces new opportunities to optimize resource provisioning by maximizing the time a certain instance type is used during a month. We defer such considerations to future work.

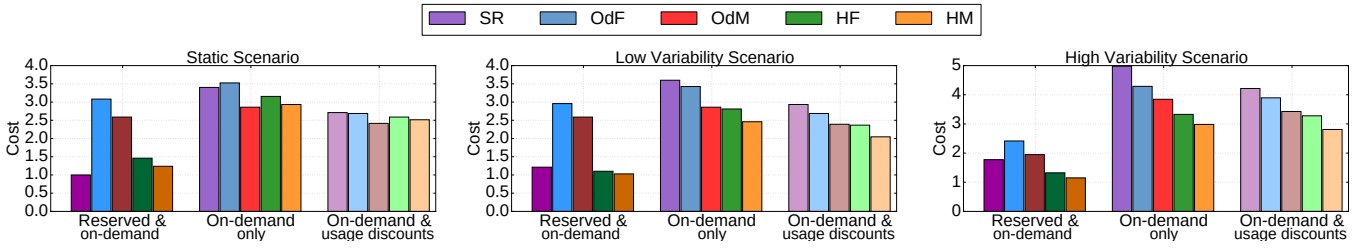


Figure 16: Sensitivity to the cloud pricing model for the three workload scenarios.

5.4. Additional Provisioning Considerations

Spot instances: Spot instances consist of unallocated resources that cloud providers make available to users through a bidding interface. Spot instances do not have availability guarantees, and may be terminated at any point in time if the bidding price is lower than the market price for an instance type. Incorporating spot instances in provisioning strategies for non-critical tasks or jobs with very relaxed performance requirements can further improve the system's cost-efficiency.

Reducing unpredictability: Resource partitioning (e.g., cache or network bandwidth partitioning) has the potential to improve isolation between instances sharing one or more resources, thus reducing performance unpredictability in fully on-demand provisioning strategies. We plan to investigate how resource partitioning complements provisioning decisions in future work.

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

6. Related Work

Cluster management: The increase in the size and number of large-scale DCs has motivated several designs for cluster management. Systems like Mesos [26], Torque [59] and Omega [54] all address the problem of allocating resources and scheduling applications 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 [54]. Dejavu identifies a few workload classes and reuses previous resource allocations for each class, to minimize reallocation overheads [60]. CloudScale [55], PRESS [22], AGILE [44] and the work by Gmach et al. [21] predict future resource needs online, often without a priori workload knowledge. Finally, auto-scaling systems, such as Rightscale [52], 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 [17,40,42,64]. Paragon uses classification techniques to determine the impact of platform heterogeneity and workload interference on an unknown, incoming workload [17]. It then uses this information to schedule each workload in a way that enables high performance for the job and high utilization for the cluster. 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 interference effects [43]. Yang et al. developed an online scheme that detects memory pressure and finds colocations that avoid interference on latency-sensitive workloads [64]. Similarly, DeepDive detects and manages interference between co-scheduled workloads in a VM environment [45]. Finally, CPI2 [68] throttles low-priority workloads that induce interference to important services. In terms of managing platform heterogeneity, Nathuji et al. [42] and Mars et al. [39] quantified its impact on conventional benchmarks and Google services, and designed schemes to predict the most appropriate server type for each 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, as well as security and privacy concerns [2,10,27,29,67]. 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. [27] 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. [67] discuss the security challenges of hybrid environments, and propose ways to leverage the private portion of the infrastructure for privacy-critical computation. The provisioning strategies discussed here are also applicable to hybrid clouds.

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 pricing strategies. Deelman et al. [15] discuss provisioning strategies for a single astronomy application on a cloud provider. Li et al. [34]

compare the resource pricing of several cloud providers to assist users provision their applications. Finally, Guevara et al. [24] and Zahed et al. [66] 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 different provisioning strategies available on cloud providers today and showed their advantages and pitfalls with respect to cost, performance predictability and initialization overheads. We have also designed two new hybrid provisioning strategies, that use both reserved and on-demand resources, and leverage the information on resource preferences of incoming jobs and the quality of previously-obtained on-demand instances, to map jobs to reserved versus on-demand resources. We showed that hybrid provisioning 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.

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 *Proc. of OSDI*. New Orleans, 1999.

[6] S. K. Barker and P. Shenoy. Empirical evaluation of latency-sensitive application performance in the cloud. In *Proc. of 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 *Proc. of IC2E*. Redwood City, CA, 2013.

[11] M. Carvalho, W. Cirne, F. Brasileiro, and J. Wilkes. Long-term slos for reclaimed cloud computing resources. In *Proc. of 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 *Proc. of SOSP*. Banff, CA, 2001.

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

[14] J. Dean and L. A. Barroso. The tail at scale. In *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 *Proc. of SC*. Austin, TX, 2008.

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

[17] C. Delimitrou and C. Kozyrakis. Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters. In *Proc. of ASPLOS*. Houston, TX, 2013.

[18] C. Delimitrou and C. Kozyrakis. Quasar: Resource-Efficient and QoS-Aware Cluster Management. In *Proc. of ASPLOS*. Salt Lake City, UT, 2014.

[19] 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 *Proc. of SOCC*. San Jose, CA, 2012.

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

[21] D. Gmach, J. Rolia, L. Cherkasova, and A. Kemper. Workload analysis and demand prediction of enterprise data center applications. In *Proc. of IISWC*. Boston, MA, 2007.

[22] Z. Gong, X. Gu, and J. Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *Proc. of CNSM*. Niagara Falls, ON, 2010.

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

[24] M. Guevara, B. Lubin, and B. Lee. Navigating heterogeneous processors with market mechanisms. In *Proc. of HPCA*. Shenzhen, China, 2013.

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

[26] 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 *Proc. of NSDI*. Boston, MA, 2011.

[27] M. Hoseinyfarahabady, H. Samani, L. Leslie, Y. C. Lee, and A. Zomaya. Handling uncertainty: Pareto-efficient bot scheduling on hybrid clouds. In *Proc. of ICPP*. Lyon, 2013.

[28] A. Iosup, N. Yigitbasi, and D. Epema. On the performance variability of production cloud services. In *Proc. of CCGRID*. Newport Beach, CA, 2011.

[29] 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.

[30] Y. E. Khamra, H. Kim, S. Jha, and M. Parashar. Exploring the performance fluctuations of hpc workloads on clouds. In *Proc. of CloudCom*. Indianapolis, IN, 2010.

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

[32] E. Lau, J. E. Miller, I. Choi, D. Yeung, S. Amarasinghe, and A. Agarwal. Multicore performance optimization using partner cores. In *Proc. of HotPar*. Berkeley, CA, 2011.

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

[34] A. Li, X. Yang, S. Kandula, and M. Zhang. Cloudcmp: Comparing public cloud providers. In *Proc. of IMC*. Melbourne, Australia, 2010.

[35] Host server cpu utilization in amazon ec2 cloud. http://huanliu.wordpress.com/2012/02/17/host-server-cpu-utilization-in-amazon-ec2-cloud/.

[36] J. Lorch and A. Smith. Reducing processor power consumption by improving processor time management in a single-user operating system. In *Proc. of MOBICOM*. New York, NY, 1996.

[37] Mahout. http://mahout.apache.org/.

[38] D. Mangot. Ec2 variability: The numbers revealed. http://tech.mangot.com/roller/dave/entry/ec2_variability_the_numbers_re%vealed.

[39] J. Mars and L. Tang. Whare-map: heterogeneity in "homogeneous" warehouse-scale computers. In *Proc. of ISCA*. Tel-Aviv, Israel, 2013.

[40] 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 *Proc. of MICRO*. Porto Alegre, Brazil, 2011.

[41] 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 *Proc. of ICCAD*. 2002.

[42] R. Nathuji, C. Isci, and E. Gorbatov. Exploiting platform heterogeneity for power efficient data centers. In *Proc. of ICAC, FL, 2007*.

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

[44] H. Nguyen, Z. Shen, X. Gu, S. Subbiah, and J. Wilkes. Agile: Elastic distributed resource scaling for infrastructure-as-a-service. In *ICAC*. 2013.

[45] D. Novakovic, N. Vasic, S. Novakovic, D. Kostic, and R. Bianchini. Deepdive: Transparently identifying and managing performance interference in virtualized environments. In *Proc. of ATC*. San Jose, CA, 2013.

[46] Openstack cloud software. http://www.openstack.org/.

[47] 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.

[48] 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.

[49] K. Ousterhout, P. Wendell, M. Zaharia, and I. Stoica. Sparrow: Distributed, low latency scheduling. In *Proc. of SOSP*. Farminton, PA, 2013.

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

[51] C. Reiss, A. Tumanov, G. Ganger, R. Katz, and M. Kozych. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proc. of SOCC*. 2012.

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

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

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

[55] Z. Shen, S. Subbiah, X. Gu, and J. Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proc. of SOCC*. Cascais, Portugal, 2011.

[56] T. Simunic and S. Boyd. Managing power consumption in networks on chips. In *Proc. of DAC*. Paris, France, 2002.

[57] 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 *Proc. of ACM SIGARCH Computer Architecture News, v.34 n.2, p.314-326, May 2006*.

[58] 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 *TACO 8(4)*. 2012.

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

[60] N. Vasić, D. Novaković, S. Miučin, D. Kostić, and R. Bianchini. Dejavu: accelerating resource allocation in virtualized environments. In *ASPLOS*. 2012.

[61] Vmware vcloud suite. http://www.vmware.com/products/vcloud.

[62] G. Wang and T. S. E. Ng. The impact of virtualization on network performance of amazon ec2 data center. In *Proc. of INFOCOM*. San Diego, CA, 2010.

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

[64] H. Yang, A. Breslow, J. Mars, and L. Tang. Bubble-flux: precise online qos management for increased utilization in warehouse scale computers. In *Proc. of ISCA*. 2013.

[65] 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 *Proc. of NSDI*. San Jose, CA, 2012.

[66] S. M. Zahed and B. C. Lee. Ref: Resource elasticity fairness with sharing incentives for multiprocessors. In *Proc. of ASPLOS*. Salt Lake City, UT, 2014.

[67] J. Y. Zhang, P. Wu, J. Zhu, H. Hu, and F. Bonomi. Privacy-preserved mobile sensing through hybrid cloud trust framework. In *Proc. of ICCC*. 2013.

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

[69] 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*.

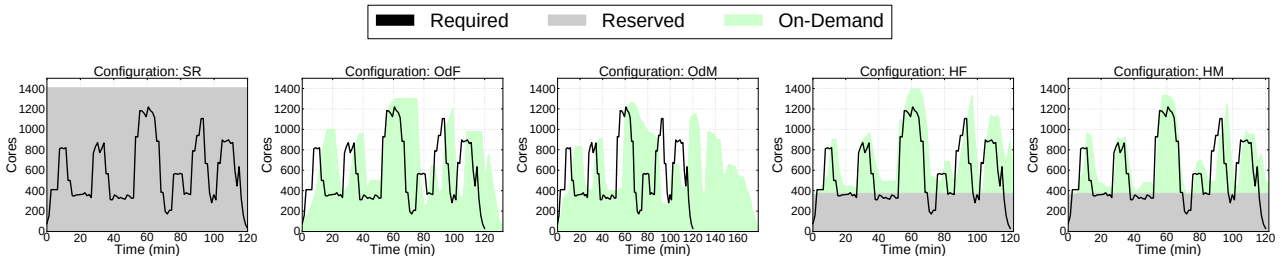


Figure 17: Resource allocation graphs for the five provisioning strategies in the case of the high variability scenario.

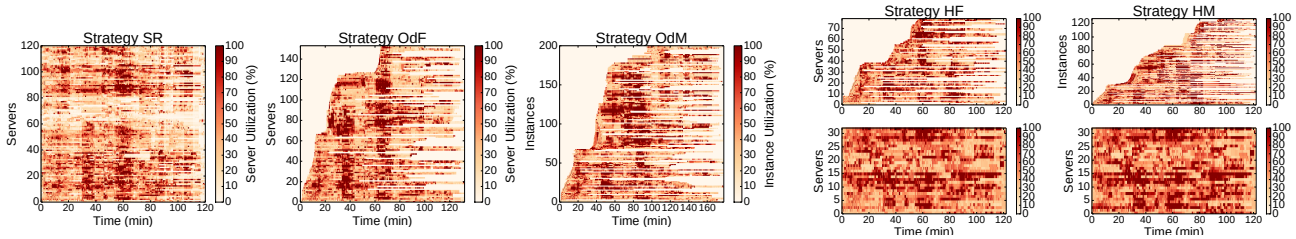


Figure 18: Resource utilization for the high variability scenario across the five provisioning strategies. For strategies HF and HM we separate reserved (bottom) from on-demand (top) resources.

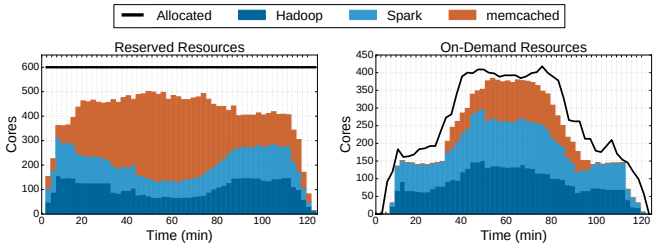


Figure 19: Breakdown of allocation per application type.

APPENDIX

We now present results that provide more in-depth understanding of the trade-offs of the different strategies.

A. Resource Efficiency

Apart from lowering cost, we also want to ensure that a provisioning strategy is not wasteful in terms of resources. Figure 17 shows the resource allocation by each strategy throughout the duration of the high variability scenario. The reserved system (SR) is provisioned statically for the peak requirements plus a 15% overprovisioning as described in Section 3.1. Because all instances are private (limited external interference) and all resources are readily available, the scenario achieves near-ideal execution time (∼2hr). However, because there is high load variability, utilization is rarely high, resulting in poor resource efficiency. Strategy OdF obtains resources as they become necessary and because it induces spin-up overheads frequently due to the constant load change, it results in longer execution time (132 min). It also introduces some overprovisioning, as it only requests the largest instances to constrain performance unpredictability. OdM does not overprovision allocations noticeably since it uses smaller instances, however, it significantly hurts performance, resulting in the scenario completing in 48% more time. Performance degradation is partially the result of variability in the quality of an instance, and of the high instance churn (releasing instances immediately after use), due to their poor behavior. 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. Spin-up overheads induce some delay in the completion of the scenario, although this only amounts to 2.6% over SR. With HM, this delay is primarily due to instances that misbehaved and were immediately released after the completion of their jobs, requiring resources to be obtained anew (about 11% of instances). This issue is less pronounced when all on-demand resources are large instances (HF).

Figure 18 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 from the on-demand resources. For the on-demand resources, instances are ranked in the order in which they are obtained.

In the case of the fully reserved provisioning strategy (SR), a small fraction of the instances operate at high utilization as we try to co-schedule as many applications as possible, however, the majority of instances are greatly underutilized. This is consistent with the findings of Figure 17a; because resources are provisioned for peak load most of the machines operate at low utilization when load is lower than the maximum. The majority of resources are used only during the two load surges at 32 and 60 minutes. Strategies OdF and OdM obtain resources as they become necessary (shown by the fact that not all instances exist at time 0). Although the total number of instances used during the execution of the scenario with OdF is similar to SR, most instances are released when no

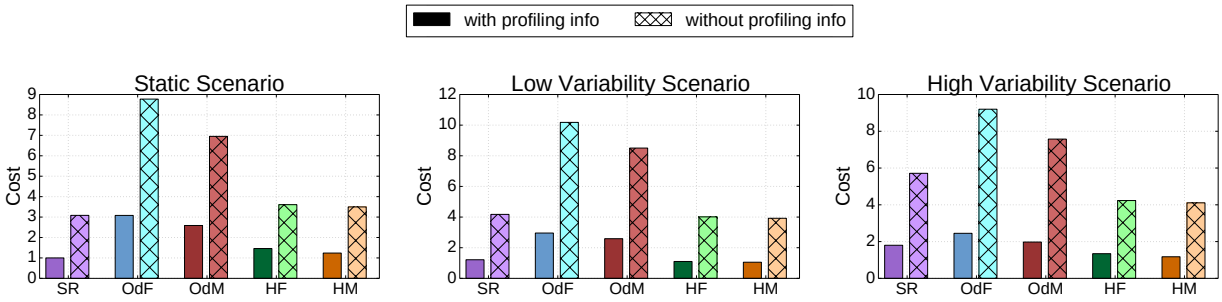


Figure 20: Cost of the three workload scenarios with and without the profiling information from Quasar.

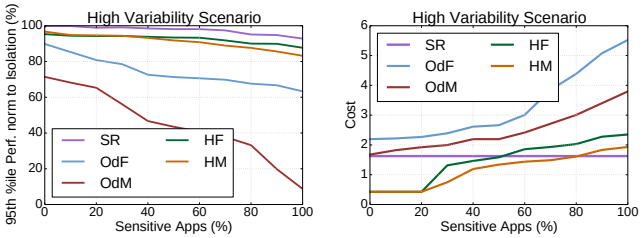


Figure 21: Sensitivity to application characteristics.

longer needed, hence the number of active instances during off-peak load is significantly lower. In the case of the OdM strategy instances are additionally released when they behaved poorly for a given application. Note that because of the high instance churn, the total number of instances used throughout the execution of the scenario is higher for OdM than for OdF. The scenario also takes longer to complete for OdM (178 as opposed to 120 minutes).

Finally the hybrid strategies maintain the utilization of reserved resources high throughout the execution of the workload scenario, and obtain on-demand resources as needed. HF needed in total 72 on-demand instances, although only 34 of those are used on average. More than 60 on-demand instances are only used during load surges. HM needs a higher number of on-demand resources, because it also uses smaller instances, and because poorly-performing instances are released and replaced by new ones. Note that the fraction of released instances due to poor performance is lower for HM than for OdM, since only jobs that can tolerate some performance unpredictability are mapped to smaller on-demand instances. Both hybrid strategies complete the scenario in the same time as SR.

Figure 19 breaks down the allocation of the low variability scenario by application type, for strategy HM. Initially the reserved resources are used for most applications, until load reaches the soft utilization limit. Beyond that, the interference-sensitive memcached occupies most of the reserved resources, while the batch workloads are mostly scheduled on the on-demand side. When the increase in the memcached load exceeds the capabilities of the reserved resources, part of the latency-critical service is scheduled on on-demand resources to avoid long queueing delays, although it is often allocated larger on-demand instances to meet its resource quality requirements.

B. Sensitivity to Workload Characteristics

We now evaluate how the performance and cost results change as the characteristics of the applications change with respect to how sensitive they are to interference. Figure 21 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 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, as it is provisioned for peak load, and there is no external load. 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 applications, both due to increased spin-up overheads, and because more applications are now affected by external 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, as increasing the fraction of sensitive applications results in a lower degree of co-scheduling and the need for new resources.

C. Cost Impact of Information from Quasar

Finally, we examine how removing the information on the resource preferences of new jobs affects the cost of the five provisioning strategies. In this case, latency-critical applica-

tions, such as memcached are provisioned for their peak load, and batch jobs (Hadoop and Spark) use the default framework parameters, for example for the number of tasks per core, heapsize, etc. Figure 20 shows the cost with and without the information from Quasar for the three workload scenarios. Since overprovisioning is now much more prominent both the statically-reserved and the on-demand strategies incur significantly higher costs. The differences become more pronounced for scenarios with load variability, where overprovisioning is higher. For most cases the relative ordering between strategies remains the same; for example in the high variability scenario even without the information from Quasar the hybrid strategies have lower cost than SR and significantly lower than the on-demand strategies.