Towards Multi-Tenant Performance SLOs

Willis Lang, Srinath Shankar, Jignesh M. Patel, and Ajay Kalhan

Abstract—As traditional and mission-critical relational database workloads migrate to the cloud in the form of Database-as-a-Service (DaaS), there is an increasing motivation to provide performance goals in Service Level Objectives (SLOs). Providing such performance goals is challenging for DaaS providers as they must balance the performance that they can deliver to tenants and the data center's operating costs. In general, aggressively aggregating tenants on each server reduces the operating costs but degrades performance for the tenants, and vice versa. In this paper, we present a framework that takes as input the tenant workloads, their performance SLOs, and the server hardware that is available to the DaaS provider, and outputs a cost-effective recipe that specifies how much hardware to provision and how to schedule the tenants on each hardware resource. We evaluate our method and show that it produces effective solutions that can reduce the costs for the DaaS provider while meeting performance goals.

Index Terms—Database management, relational databases

1 Introduction

Traditional relational database workloads are quickly moving to the cloud in the form of Database-as-a-Service (DaaS). Such cloud deployments are projected to surpass the "on-premises" market by 2014 [33]. As this move to the cloud accelerates, increasing numbers of mission-critical workloads will also move to the cloud, and in turn will demand that the cloud service provider furnish some assurances on meeting certain quality-of-service metrics. Some of these metrics, such as uptime/availability, have been widely adopted by DaaS providers as Service Level Objective (SLOs) [4], [39]. (SLOs are specific objectives that are specified in the encompassing Service Level Agreement, a.k.a. SLA.) Unfortunately, performance-based SLOs have still not been widely adopted in DaaS SLAs. Performance-based SLOs have been proposed in other (non-DaaS) cloud settings [22], and in the near future it is likely that DaaS users will demand these SLOs (especially if they are running mission-critical database applications that require a certain level of performance). DaaS providers may also provide performance-based SLOs as a way to differentiate their services from their competitors.

DaaS providers want to promise high performance to their tenants, but this goal can often conflict with the goal of minimizing the overall operating costs. Data centers that house database services can have high fixed monthly costs that impact the DaaS providers' bottom line [15], [21]. For a DaaS provider, servicing the same tenants with fewer servers decreases the amortized monthly costs [37]. Hence, consolidation via *multi-tenancy* (where multiple database tenants are run on the same physical server) is a straightforward way to increase the cost-effectiveness of the DaaS deployment.

In a traditional single tenant database setting, two key factors that determine performance are: a) The workload characteristics; and b) The server hardware on which the database management system (DBMS) is being run. In a multi-tenant setting, the degree of multi-tenancy becomes an additional factor that impacts performance, both for the overall system and the performance that is experienced by each individual tenant. In general, increasing the degree of multi-tenancy decreases per-tenant performance, but reduces the overall operating cost for the DaaS provider.

Hence, the important question for a DaaS provider is how to balance multi-tenancy with performance-based SLOs. The focus of this paper is on posing this question and presenting an initial answer. We fully acknowledge that there are many open questions that need to be answered beyond our work here, which points to a rich direction of future work.

In this paper, we propose a general DaaS provisioning and scheduling framework that optimizes for operating costs while adhering to desired performance-based SLOs. Developing a framework to optimize DBMS clusters for performance-based SLOs is challenging because of a number of specific issues, namely: (a) The DaaS provider may have a number of different hardware SKUs (Stock Keeping Units) to choose from, and needs to know how many machines of each SKU to provision for a given set of tenants – thus the provider needs *a hardware provisioning policy*; and (b) The DaaS provider also needs to know an efficient mapping of the tenants to the provisioned SKUs that meets the SLOs for each tenant while minimizing the overall cost of provisioning the SKUs – thus the DaaS provider needs a *tenant scheduling policy*. Note that the tenants on the same server may have different performance requirements,

- *W. Lang and J. M. Patel are with Computer Sciences Department, University of Wisconsin-Madison, Madison, WI 53706 USA. E-mail: {wlang, jignesh}@cs.wisc.edu.*
- *S. Shankar is with Microsoft Gray Systems Lab, Madison, WI 53719 USA. E-mail: srinaths@microsoft.com.*
- *A. Kalhan is with SQL Azure, Microsoft Corp., Redmond, WA 98052 USA. E-mail: ajayk@microsoft.com.*

Manuscript received 8 May 2012; revised 26 Apr. 2013; accepted 29 Apr. 2013. Date of publication 2 May 2013; date of current version 29 May 2014.
Recommended for acceptance by J. Gehrke, B.C. Ooi, and E. Pitoura.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier 10.1109/TKDE.2013.74

and the tenants may interfere with each other, making the mapping of tenants to the SKUs challenging.

Let us consider a concrete example to illustrate these issues. Assume that a DaaS provider has many tenants that have workloads that are like TPC-C scale factor 10. The performance metric that is of interest here is transactions per second (tps). Assume that the DaaS provider has 10,000 tenants split into two classes: '**H**' and '**L**'. The tenants in the **H** class are associated with a high performance SLO of 100 tps, whereas the tenants in the **L** class are associated with a lower performance SLO of 10 tps (and presumably a lower price). Assume that 20% of the tenants (2000 tenants) belong to the class H and the remaining (8000 tenants) belong to the class L. For this example, imagine that there is only one SKU, and assume that all the tenants have the same query workload characteristics (i.e., all tenants have the same query workload, and issue queries to the server with the same frequency).

To find a hardware provisioning policy and the associated tenant scheduling policy, we first need to understand how the performance of the tenants in class H (and class L) changes for a workload that consists of a mix of these tenants. In other words, we need to characterize the performance that *each* tenant sees for varying mixes of tenants from the two classes, when these tenants are scheduled on the same server. We capture this performance trait in a *SKU performance characterizing model*.

To produce the SKU performance characterizing model, we first benchmark the server SKU for a *homogeneous* mix of tenants. This benchmark shows that we can accommodate around 25 tenants of class H (100 tps). Scheduling more than 25 tenants results in the tps dropping below 100 tps, and hence breaks the performance SLO. Similarly, we find that this SKU can accommodate up to 100 tenants of class L (10 tps). Points A and B in Fig. 1 correspond to the findings from this homogeneous benchmark. (Below we describe what Fig. 1 shows in more detail.)

The homogeneous benchmark above defines the boundaries of how many tenants of each class we can pack on a given server. Next, we need to characterize the space to allow for an arbitrary mix of tenants. We note that while it is possible that an optimal hardware provisioning policy and associated tenant scheduling policy could only have SKUs with homogeneous tenants (i.e., no SKU has a mix of tenants from the two classes), it is also possible that the optimal policy has a mix of tenants from the two classes on some or all the SKUs. This may be the case if different tenant workloads have different resource utilizations (memory vs. disk vs. CPU) on a SKU. Thus, the SKU performance characterizing model must also consider *heterogeneous* mixes of tenants.

To complete the SKU performance characterizing model, we need to benchmark the server for varying mixes of tenants from the two performance classes, and measure the throughput that each tenant in each class sees. Fig. 1 shows the SKU performance characterizing model for an actual SSD-based server SKU using experimental results for the 100 tps and the 10 tps TPC-C tenant classes. (See Section 2 for details.)

In Fig. 1, the performance of the class H tenants is shown in Fig. 1(a), while the performance that the class L tenants experience is shown in Fig. 1(b).

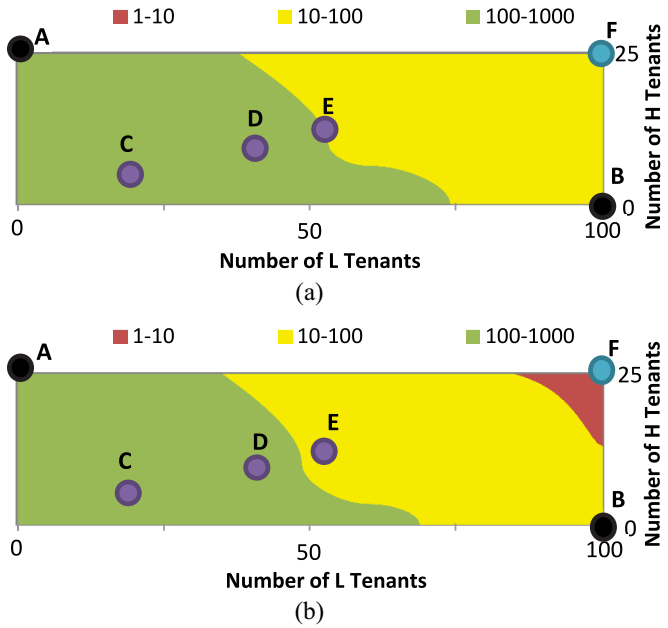

Fig. 1. Average performance seen by tenants in class H (100tps) and class L (10tps) on TPC-C scale factor 10 database as the tenant mix is varied. In both figures, circles annotated with the same letter correspond to the same operating point: (a) Performance (tps) for class H tenants. (b) Performance (tps) for class L tenants.

First, consider a homogeneous tenant scheduling policy that uses only the points A (25 100tps tenants), and B (100 10tps tenants). In this case, the DaaS provider needs to provision 160 SSD-based servers for the 10,000 tenants (80 for the H class tenants, and 80 for the L class tenants).

But, could we do better than using a homogeneous tenant scheduling policy? To answer this question, we need to systematically explore the entire space of tenant workload mixes, and the associated hardware provisioning (to compute the operating cost). Essentially, we need to explore the entire space shown in Fig. 1. Note that some of the points in this space are not feasible "solutions", as they violate the performance SLOs. For example, at the operating point F in Fig. 1(a), the H class tenants see a performance level that is below 100 tps, since the point F is in the yellow zone that corresponds to 10-100 tps. In Fig. 1(b), at point F, the L class tenants do not reach a satisfactory performance either.

On the other hand, in Fig. 1, the operating points C, D, and E are all feasible, but they result in different hardware provisioning policies, which in turn impacts the overall operating costs. In this case, the operating point E is the most cost-effective of these three operating points, because it only requires 143 SKUs (14 H tenants and 56 L tenants per SKU). In contrast, the operating point D (10 H and 40 L tenants per SKU) and the operating point C (5 H and 20 L tenants per SKU) require 200 and 400 SKUs respectively. Notice that the policy from point E results in 17 fewer servers required than the homogeneous policy from point A and B.

The problem illustrated above becomes even more complicated if the DaaS provider has a mix of SKUs to choose from. In this case, assume that the DaaS provider has another SKU that is cheaper, but has lower overall performance on this workload. In this case, the DaaS provider

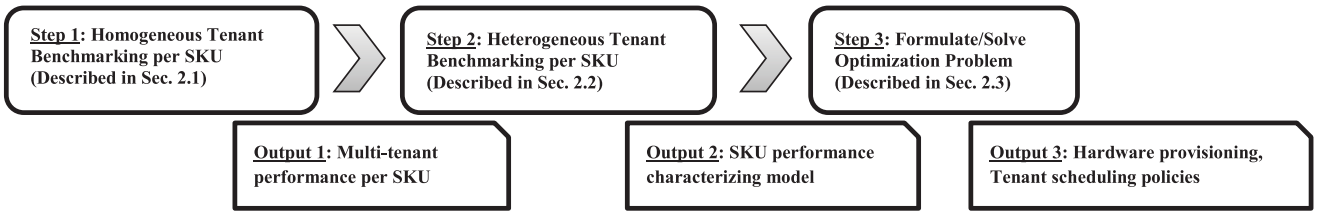

Fig. 2. Work-flow diagram for using our framework.

needs to consider the cost ratio between the two different SKUs and the relative performance differences, and provision hardware that reduces the overall operating cost. Note that the lowest cost feasible operating point could involve deploying a mix of the two (or, in general, more) SKUs, as shown by various examples in Section 3. Thus, the overall optimization problem involves finding a mix of SKUs to deploy for a given set of tenants belonging to different performance-based SLO classes, along with a tenant scheduling policy for each deployed SKU. In this paper we present and evaluate a solution to this problem.

This paper makes the following contributions:

- To the best of our knowledge, our work presented in [30] and this extended version[1] are the first papers to formulate and explore the problem of how to provision servers in a DaaS environment with the goal of providing performance-based SLOs.
- We develop an optimization framework to address the problem above. This framework outputs an SLO compliant tenant scheduling strategy and a cost-minimizing hardware provisioning strategy that together serve as the recipe for deploying resources and operating the DaaS for the input workload.
- We evaluate our method and demonstrate the effectiveness of our approach.

The remainder of this paper is organized as follows: We present our framework in Section 2 and an applications case-study in Section 3. We study dynamic cloud environments in Section 4 and a discussion follows in Section 5. Related work is discussed in Section 6 and our concluding remarks follow in Section 7.

2 Performance SLO Framework

In this section, we describe our optimization framework, which has three steps as shown in Fig. 2. Recall that the goal of this framework is to provide hardware provisioning and tenant scheduling policies that minimize the costs to DaaS providers while satisfying the performance-related specifications in tenant SLOs.

In the first step in Fig. 2 (described in Section 2.1), we benchmark the performance of each server SKU in a *homogeneous* multi-tenant environment. At the end of this step, we understand the tenant performance for each tenant class on each hardware SKU, producing Output 1 in Fig. 2. From this first step, for a specific performance level, we can determine the maximum number of tenants of a given class that can be scheduled on a specific server SKU, such that the performance SLOs can be satisfied for each tenant. Essentially, in this step we find points like A and B in Fig. 1 for every tenant class for every hardware SKU.

The next step, marked as Step 2 in Fig. 2, uses Output 1 to compute the boundaries of the space of mixed class workloads that should be considered. Then, for each hardware SKU this space is characterized by running actual benchmarks. In other words, Step 2 computes Fig. 1 for every hardware SKU as Output 2. Now, we understand the impact of scheduling a workload with tenants that have different SLO requirements on the same server box. This step is discussed in more detail in Section 2.2.

The last step in Fig. 2 takes as input the set of SKU performance characterizing models (i.e., Output 2) and computes an optimal strategy to deploy the workload. This step uses an optimization method that takes as input (i) A set of performance SLOs; (ii) A set of hardware SKUs with specific costs and performance characteristics; (iii) A set of tenants with different performance SLOs to be scheduled; and computes the hardware provisioning and the tenant scheduling policies that minimize costs while satisfying all SLOs. This step is discussed in more detail in Section 2.3.

2.1 Characterizing Multi-Tenant Performance

This section discusses the first step in our framework that is shown in Fig. 2.

2.1.1 Workload and Performance Metric

To make the discussion concrete, in this paper we use TPC-C as a model workload, which has also been used before to study DaaS [18].

Each of our TPC-C transactions were implemented as stored procedures within SQL Server. Our application driver issued stored procedure calls to SQL Server via .NET connections from network-attached clients. Like prior studies [24], we maintained the full transaction mix ratio as dictated by TPC but eliminated think-time pauses, implemented each tenant with a single remote application driver, and did not scale the number of clients with warehouses. As a performance metric, we use the throughput of the *new-order* transactions, as is done for reporting TPC-C results[2].

2.1.2 Hardware SKUs

Table 1 shows the two server SKUs, *ssdC* and *diskC*, that we use in this paper. Both servers are identical except for the

1. This extended version includes an analysis of the cost to maintain globally optimal solutions, and the cost of tenant SLO elasticity.

2. Disclaimer: While we have used the TPC-C benchmark as a representative workload in this paper, the results presented are not audited or official results, and, in fact, were not run in a way that meets all of the benchmark requirements. Consequently, these results should not be used as a basis to determine SQL Server's performance on this or any other related benchmark.

TABLE 1
Two Server Configurations (SKUs)

	ssdC	diskC
CPU	2X Intel L5630	2X Intel L5630
RAM	32GB DDR3	32GB DDR3
OS Storage	10K SAS 300GB	10K SAS 300GB
DB Data	2X Crucial C300 256GB	2X 10K SAS 300GB
DB Log	Crucial C300 256GB	10K SAS 300GB
RAID Cntlr w/BBC	YES	YES
Cost	$4,500	$4,000

storage subsystem. Both server SKUs are configured with low-power Nehalem-based L5630 Intel processors (dual quad cores), and 32GB DDR3 memory, running Windows Server 2008R2 and the latest internal version of SQL Server. The OS and the DBMS are installed on a separate 10K RPM 300GB SAS drive. In the *ssdC* configuration, all the data files and log files of the database are stored on three Crucial C300 256GB SSDs while in the *diskC* configuration, these are stored on three 10K RPM 300GB SAS drives.

We note that the storage subsystem has a big impact on the RDBMS performance in a multi-tenant environment, since the load imposed on the hardware when serving independent tenant requests naturally leads to randomized data access. This behavior is in contrast to traditional single-tenant environments where the DBMS schedules data accesses to be as sequential as possible.

2.1.3 Multi-Tenancy and Performance

There are many ways to deploy a DaaS on a cluster with multi-tenancy [5], [6], [16], [18], [33]. We list four main approaches to housing tenants that have emerged recently in decreasing order of complexity: (1) all tenant data are stored together within the same database and the same tables with extra annotation such as 'TenantID' to differentiate the records from different tenants [5], [6]; (2) tenants are housed within a single database, but with separate schemas to differentiate their tables and provide better schema-level security; (3) each tenant is housed in a separate database within the same DBMS instance (for even greater security); (4) each tenant has a separate Virtual Machine (VM) with an OS and DBMS, which allows for resource control via VM management [18].

We use option 3 to implement multi-tenancy, since this option provides a good trade-off between wasted resources due to extra OSs in the VM method (option 4), and the complex manageability and security issues associated with options 1 and 2 [18]. Looking at the other options is an interesting direction for future work.

In our experiments, we consider a workload comprised of 1GB TPC-C tenants with 10 warehouses. We recorded the average per-tenant TPC-C transactions per second achieved on both hardware SKUs for varying degrees of multi-tenancy over a timespan of 100s. These results are shown in Fig. 3.

There are a few important observations from Fig. 3. First, on our hardware SKUs, the only way tenants can achieve a performance of 100tps is if their datasets almost completely fit in memory. Note the drop-off in tps when the number of tenants is increased beyond 25 (i.e., after the combined tenant size crosses 25*GB*). Second, when the datasets fit

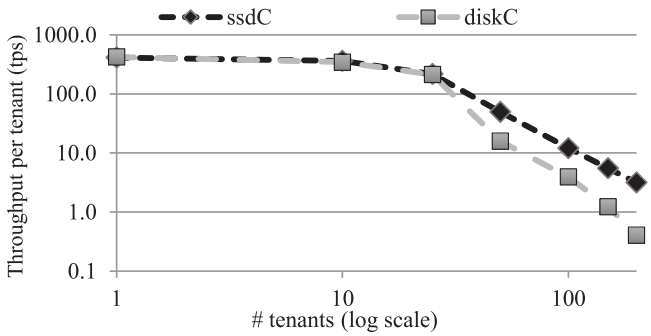

Fig. 3. Performance for the ssdC and diskC SKUs (see Table 1) as we increase the number of tenants on a single SKU.

completely in memory, the cheaper *diskC* server can deliver the same per-tenant performance as the more expensive *ssdC* server since the storage subsystem is not the bottleneck. Finally, notice that at the lower performance levels, the *ssdC* server can support significantly more concurrent tenants than the *diskC* server. This behavior is due to the better random I/O performance of the SSD storage compared to the mechanical disk storage. For instance, in Fig. 3, the measured log disk utilization at 10 tenants for the *ssdC* and *diskC* SKUs was 39% and 41% respectively. As we increased the number of tenants to 25, the log disk utilization increased to 50% and 66% for these two SKUs respectively. Finally, at 50 tenants and beyond, the log disk utilization is saturated at more than 95% for both SKUs.

The curve shown in Fig. 3 defines the maximum number of tenants that each SKU can support while maintaining a specific performance level per tenant. This homogeneous multi-tenant benchmarking is a necessary first step since it defines the boundaries of the performance that the DaaS provider can promise in their SLOs.

Definition 1. *Let the set* $S = \{s_1, s_2, \ldots, s_k\}$ *represent the* k *SLOs published by a DaaS provider.*

Typically, $k > 1$ since different tenants may require (and be willing to pay for) different levels of performance. Given a set of tenants with different SLOs to schedule on a cluster, a natural scheduling policy is to schedule the tenants of each class on the type of server that can handle the most number of tenants of that class. However, this approach ignores the relative cost of different SKUs, as well as the possibility of scheduling tenants of different classes on the same server to reduce the overall provisioning and operating costs. The next step (Section 2.2) is to determine the behavior of a single SKU when loaded with tenants that are associated with different SLOs.

2.2 Characterizing Heterogeneous SLOs

A number of mechanisms can be used to provide different performance SLOs on the same server. One simple mechanism is resource governance whereby tenants are allocated specific amounts of critical resources like CPU and DBMS buffer pages to limit their resource consumption. Another mechanism is to use an admission control server that throttles incoming tenant requests accordingly. Studying the different mechanisms to implement performance SLOs is an interesting topic, but is orthogonal to our optimization framework, and hence beyond the scope of this paper.

To avoid the additional complexity of an admission control server, we chose to simulate a buffer pool resource governance mechanism on top of SQL Server. In our method, we start separate SQL Server instances within each physical server with one instance for each SLO class s_i (there are k of these as per Definition 1). All tenants that belong to the same SLO class s_i are assigned to the same SQL Server instance. The performance of each SQL Server instance is throttled by limiting the amount of main memory that is allocated to it. The amount of main memory that is allocated to each SQL Server instance (SLO class) is an average of two factors. The first factor is the fraction of the tps requirements for that SLO class compared to the aggregate total tps across all the SLO classes. The second factor is the ratio of tenants in that SLO class to the total number of tenants. This memory allocation method provides a balance between allocating memory purely based on tps and purely based on the number of tenants. (We experimented with other methods, but found that this method provided the best overall behavior allowing us to pack far more tenants per SKU than other simpler methods. In the interest of space we omit these additional details.)

Recall that Fig. 3 characterizes the performance of the server SKUs *ssdC* and *diskC* when all the tenants on a SKU have equal access to resources. Given tenants with different SLOs (Definition 1), we need to characterize the performance delivered by each server SKU to each tenant class s_i. For this purpose, we use a SKU performance characterizing function, which is described next.

Definition 2. *For a given SKU, let $\vec{b} = [b_1\ b_2\ \ldots\ b_k]^T$ where b_i represents the number of tenants of class s_i scheduled on the server. For this server, the SKU performance characterizing function, $f(\vec{b})$, represents the performance delivered over a specific time interval for different tenant scheduling policies. Here $f(\vec{b}) = [\phi_1\ \phi_2\ \ldots\ \phi_k]^T$ where ϕ_i is the random variable representing the performance achieved by the tenants of class s_i scheduled on the server.*

Using this definition for function f, it is possible to provide the performance SLOs in the same way as the current uptime SLAs. For instance, say that for a given SKU with a load defined by $\vec{b}$, we determine that the distribution of the measured performance over 100 seconds for the tenants of class s_i (say, a 100tps class) is normal, with an average of 130 tps and a standard deviation of 10tps; that is, $\phi_i \sim N(130, 10)$. Then, according to the definition of a normal distribution, for all the 100tps tenants that are scheduled on this server, we can guarantee the desired performance 99.6% of the time.

The ability to provide such guarantees makes our formulation of the SKU characterizing function f very powerful in defining performance SLOs. In practice, fully characterizing f is likely to be challenging and one has to simplify this function. In this paper, we consider the following simplification of f to a boolean characterizing function (exploring other options is an interesting direction for future work).

Definition 3. *Given a certain server SKU and $\vec{b}$ from Definition 2, a simplified boolean SKU performance characterizing function $\hat{f}(\vec{b})$ returns true if all the tenants achieve their respective SLO performance based on a set of summary statistics of the random variables and false otherwise.*

As a simplification for our experiments, we ignored other statistics such as variance and defined $\hat{f}(\vec{b})$ in terms of the average transactions per second over 100s. For example, consider Fig. 1, we plotted $f(\vec{b}) = [E[\phi_1]\ E[\phi_2]\ \ldots\ E[\phi_k]]^T$ for *ssdC* (see Table 1) for two SLO classes, $S = \{100tps, 10tps\}$.

Having defined the SKU performance characterizing function, the next question is to find acceptable operating zones that deliver the promised performance to each tenant in each class s_i. Again, using Fig. 1 as an example, we wish to compute the area in both sub-figures where both the 100tps tenants and the 10tps tenants meet their performance requirements. This area defines the acceptable "operating zone" for the *ssdC* SKU, and is distinguished from the other areas using Definition 3.

To evaluate the function $\hat{f}$, a systematic search of the tenant scheduling space is performed as follows: We first start by scheduling the maximum number of highest-performance tenants as determined by the benchmarking step in Section 2.1. Then, we systematically substitute a fixed small number of these highest-performance tenants with low-performance tenants (if there are more than two tenant classes, in this step, we can iterate through fixed size combinations of the lower performance tenant classes). For each sample, we run a benchmark with the current mix of tenants, and record the observed per-tenant performance. If the observed performance satisfies all tenant SLOs, then $\hat{f}$ returns true for this tenant scheduling policy and for all other scheduling policies where there are fewer tenants in any of the classes. If $\hat{f}$ returns true, we also try adding more low-performance tenants (iteratively in every low performance class) and repeat the experiment. We keep pushing up the number of the tenants in the low performing tenant class(es) until $\hat{f}$ returns false, in which case we know we have reached the boundary of the $\hat{f}$ function. Thus, we determine a tenant scheduling "frontier", so that $\hat{f}$ is true on one side of the frontier and false on the other side. (As part of future work, it would be interesting to consider obtaining this frontier via other methods such as augmenting the query optimizer module to generate/estimate this frontier [12], [19].)

2.2.1 Frontier for the SLO mix – 10tps and 1tps

Consider the SLO set $S = \{10tps, 1tps\}$, and the SKUs ssdC and diskC (see Table 1). The frontiers for this case are shown in Fig. 4 as the solid black line. The diamond points in this figure represent some of the actual benchmark tests that were run. The points that lie *above* a frontier line represent tenant scheduling policies that *fail* to meet tenant SLOs ($\hat{f} = false$), whereas the points that lie *on* the frontier line will satisfy all tenant SLOs. The area below the frontier line contains scheduling policies that will satisfy tenant SLOs but potentially waste resources (i.e., are potentially over-provisioned).

An interesting point about the performance characteristics shown in Fig. 4 is that the bottleneck for the points in the frontier is the log disk. Each database has a log file and as more tenants are added, the I/Os to the log

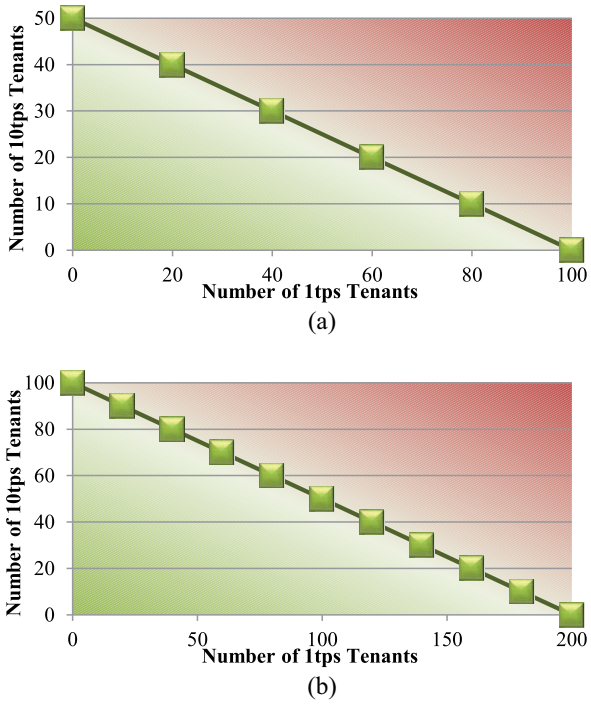

Fig. 4. SKU performance characterizing functions for $S = \{10tps, 1tps\}$: (a) Performance on the diskC SKU. (b) Performance on the ssdC SKU.

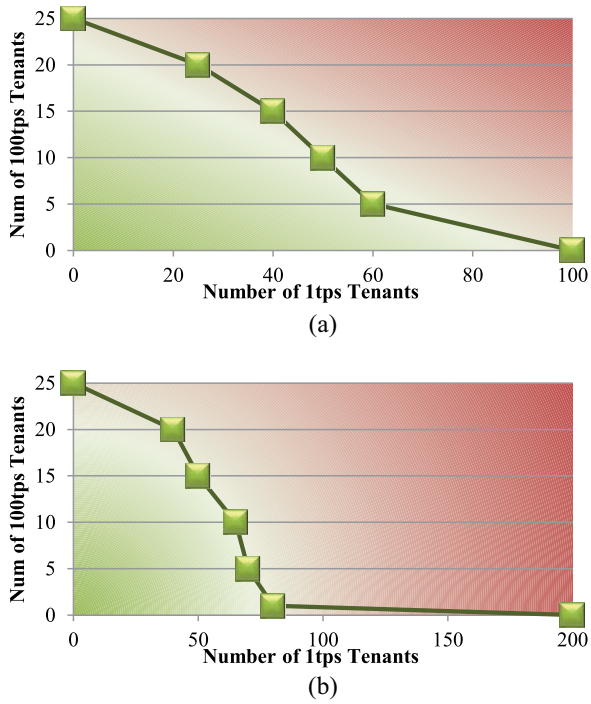

Fig. 5. SKU performance characterizing functions for $S = \{100tps, 1tps\}$: (a) Performance on the diskC SKU. (b) Performance on the ssdC SKU.

disk become more random, and each log I/O becomes relatively more expensive. As a result, if we look at the pure 10tps case (upper left point in the graph) and remove x of the 10tps tenants, we can add far fewer than $10x$ 1tps tenants.

Having a linear frontier as is the case in Fig. 4 implies that we can add/remove tenants of different classes to a server according to a constant ratio. For example, consider again the frontier for the *diskC* SKU (Fig. 4(a)) and the *ssdC* SKU (Fig. 4(b)). The slope of the lines in both graphs is $-\frac{1}{2}$, which implies that for any operating point along these two frontier lines, the DaaS provider can safely swap one 10tps tenant for two 1tps tenants. Thus, a linear frontier simplifies the tenant scheduling policies. As we discuss below, we may not always observe a linear frontier.

2.2.2 Frontier for the SLO mix – 100tps and 1tps

Suppose that a DaaS provider wishes to publish a 100tps SLO. From Fig. 3, we know that for both SKUs, we are limited to about 25 100tps tenants on either SKU. Fig. 5(a) and (b) show the observed frontiers for both the *diskC* and *ssdC* SKUs respectively, for $S = \{100tps, 1tps\}$. The frontiers are no longer linear and show that if we start from the case of only 100tps tenants (upper left point in both graphs), the initial curve is convex and then tapers off into a concave shape. At the "only 100tps tenants" point, the system is memory bound (see Fig. 3). As we move to the right along the frontier, the system now becomes log disk bound.

The initial shape of the frontier is convex since the log disk saturates a little beyond the proportions dictated by the line formed by connecting the two end points of the frontiers. For example, in Fig. 5(a) as we move from the 25 100tps case to the right, we reach a point where there are 20 100tps tenants. If the frontier were linear, then we should only be able to add $5 \times 4 = 20$ 1 tps tenants, but we can add 25 1tps tenants before the log disk saturates.

Now consider the concave tail of the frontier in Fig. 5. Again this has to do with the log disk. Consider the (bottom) right-most point in the frontier. Here we have only 1 tps tenants. At this point, the system is bottlenecked on the log disk. This behavior is captured in Fig. 6, which plots the log disk performance (y axis) of an *ssdC* server with one 100tps tenant as the number of 1tps tenants is varied (x axis). The log write wait time is shown as a range by a vertical bar where the low point denotes the first quartile and the high point denotes the third quartile. The horizontal (green) bar denotes the average. The performance achieved by the 100tps tenant (shown on the right vertical axis) is plotted using round dots.

In Fig. 6, we see that at the 200/0 point, the log disk writes takes an average of 12 ms (and the log disk is saturated at this point). If we move to the left from this point by dropping 25 1 tps tenants and adding one 100 tps tenant, then the 100tps tenant only achieves around 20 tps. As we continuously decrease the number of 1tps tenants by 25, we observe that the average log write wait time decreases only after 125 1tps tenants. The performance achieved by the 100tps tenant very closely follows with a jump at 100 1tps tenants. These results show why scheduling one 100tps tenant onto the server in Fig. 5 requires a substantial drop in 1tps tenants. To summarize, *a high performance tenant requires disproportionately large headroom in log disk provisioning to process transactions with a high throughput. Thus, even though the tenants are all running the same workload, the sheer increased performance requirement of some tenants over others causes resource*

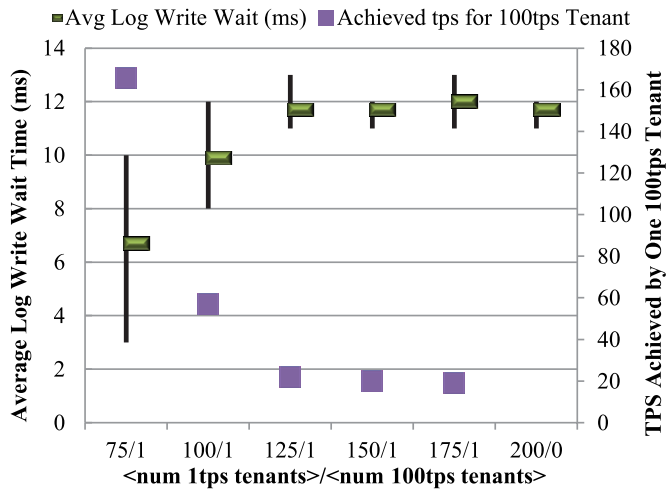

Fig. 6. Average database log write wait time with vertical bars spanning the 1st to the 3rd quartiles, along with the average tps achievable by a single 100tps tenant on the *ssdC* SKU.

requirement disparities similar to tenants running different workloads.

2.2.3 Frontier for the SLO mix – 100tps and 10tps

Now let us consider a mix of 100tps and 10tps tenants, i.e., $S = \{100tps, 10tps\}$. The results for this case are shown in Fig. 7(a) and (b) for the diskC and the ssdC SKUs respectively. For the same reasons as discussed in Section 2.2.2, we observe a knee near the lower right corner of the frontier line, and a convex shape near the upper left corner of the frontier line.

2.3 Step 3: Putting It All Together

The previous section described how to compute the SKU performance characterizing function for each SKU. We can now use these functions to formulate and solve the optimization problem for provisioning hardware and scheduling tenants that satisfy different performance SLOs (namely Step 3 in Fig. 2).

Definition 4. *M is a multiset $\{m_1, m_2, \ldots, m_p\}$ where each m_j represents a server SKU defined by a pair $m_j = (\hat{f}_j, c_j)$ where function $\hat{f}_j$ is the simplified SKU characterizing function (defined in Definition 3) and c_j represents the amortized monthly operating cost for a server.*

Note that since M is a multiset, m_j need not be unique. This allows a single server SKU to be scheduled with tenants in different ways.

Recall that we have the set of published SLOs as defined in Definition 1. We must now associate each tenant with its corresponding SLO.

Definition 5. *Let t_i represent the set of tenants that subscribe to SLO s_i as defined in Definition 1. We represent all tenants by the set $T = \cup_{i=1}^{k} t_i$.*

Using Definitions 1 to 5, the following definition describes the main optimization (minimization) problem.

Problem Definition 1. *Given the sets S, T, and multiset M, compute $a = [\alpha_1\ \alpha_2\ \ldots\ \alpha_p]$ and $B = [\vec{b_1}\ \vec{b_2}\ \ldots\ \vec{b_p}]^T$, where α_i is the needed number of servers of type m_i, and $\vec{b_i}$ is a vector of length k indicating how many tenants of each of the k SLO classes should be scheduled on an individual server of type m_i. The objective function $C = \Sigma_{i=1}^{p} \alpha_i c_i$ satisfies the following constraints:*
Constraint 1 : $aB = [|t_1|\ |t_2|\ \ldots\ |t_k|]$ (cover all the tenants)
Constraint 2 : $\hat{f}_i(\vec{b_i})$ returns true for $1 \le i \le p$ (all SLOs are satisfied)

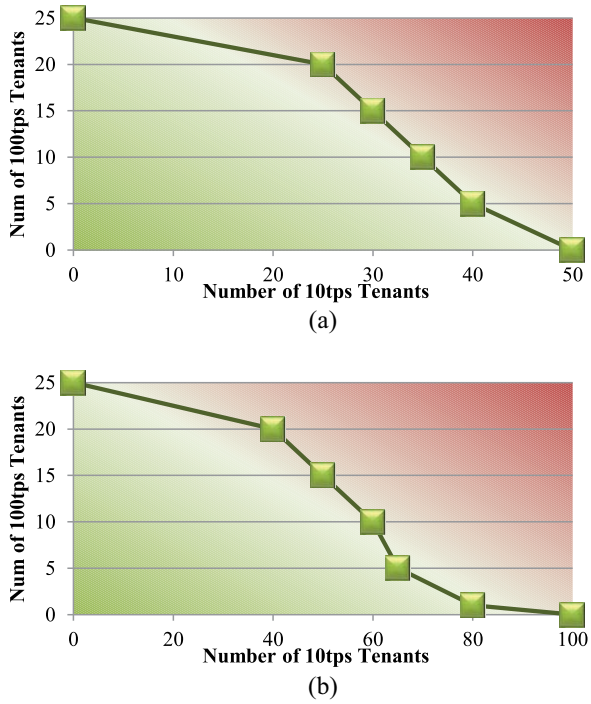

Fig. 7. SKU performance characterizing functions for $S = \{100tps, 10tps\}$: (a) Performance on the diskC SKU. (b) performance on the ssdC SKU.

Problem Definition 1 is a non-linear programming problem in the general case.[3] Here, we need to compute the following variables:

1) a – the number of servers used for each SKU. This vector determines the total cost for provisioning the servers.
2) B – the tenant scheduling policy.

The entire space of solutions does not need to be fully explored since the feasible regions are defined by the $\hat{f}$ characterizing functions and the curves defined by Constraint 1 of Problem Statement 1. Since our solution space is relatively small, a brute-force solver that explores the non-negative integer space bounded by these curves sufficed for our purposes.[4] Exploring other approaches is part of future work.

With this brute-force solver and the experimental results from Section 2.2, we now have the tools that we need to evaluate our framework.

3. In simple cases, we can parameterize the problem into a linear programming problem, but this is increasingly onerous when faced with non-linear piecewise frontier functions that characterize the server SKUs. The approach we take to solving the non-linear programming problem is much more straight-forward.

4. For a 5000 100tps tenant and 5000 10tps tenant problem, our 1 thread, brute-force solver finds a solution within 80s. on a 2.67Ghz Intel i7 CPU.

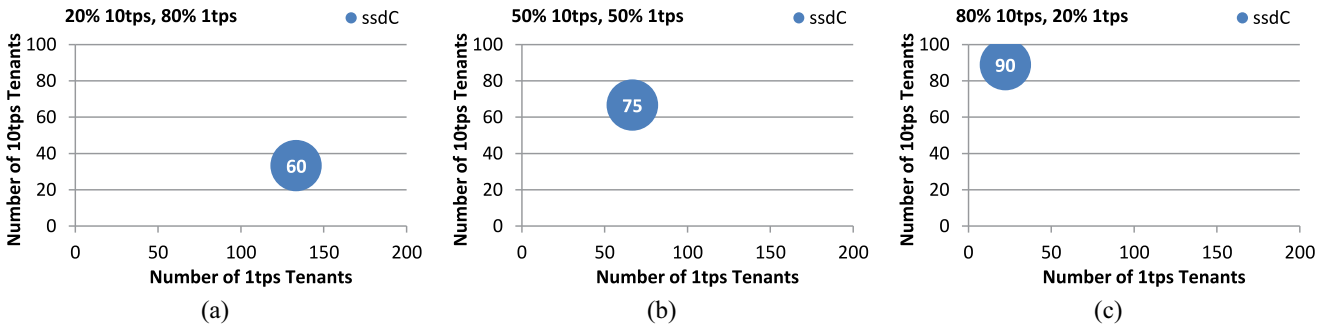

Fig. 8. Solutions for (a) SC1 - $7,500; (b) SC2 - $9,375; (c) SC3 - $11,250 (see Table 2 for details). Circle positions indicate tenant scheduling policy and circle size/annotation indicate hardware SKU provisioning policy.

3 Applications of the Framework

In this section we apply the framework described in Section 2 to hypothetical DaaS scenarios to illustrate the merits of the hardware provisioning and tenant scheduling policies described above.

In our evaluation, we assume that the DaaS provider must accommodate a total of 10,000 tenants running TPC-C scale 10 workloads, with two available SKUs – *ssdC* and *diskC* – as described in Section 2.1.2. We varied the following three parameters to arrive at the 18 scenarios listed in Table 2.

1) Published set of SLOs: We limited ourselves to three sets of SLOs discussed in Section 2.2, namely $S_1 = \{10tps, 1tps\}$, $S_2 = \{100tps, 1tps\}$, and $S_3 = \{100tps, 10tps\}$. We used average tps over 100s as the metric to determine if an SLO is satisfied or not.
2) Tenant ratios: For each SLO set S_i, we varied the relative proportion of tenants belonging to one SLO versus the other. We used three ratios in our scenarios – 20:80, 50:50 and 80:20. For instance, a 20:80 ratio for the SLO set $\{100tps, 1tps\}$ means that 2000 tenants are associated with the 100tps SLO while 8000 tenants are associated with the 1tps SLO.
3) Relative costs between server SKUs: The true purchase costs of a single *ssdC* and *diskC* server are $4,500 and $4,000 respectively. Amortized over 36 months [21], we arrived at monthly costs of $125 and $111 respectively. Although in reality the *diskC* server is 10% cheaper than *ssdC*, we also considered a hypothetical *diskC* price point of $3,150 ($88 amortized, 30% less than *ssdC*) to consider what happens if the relative costs of the hard disks were lower (e.g., if we had used cheaper SATA3 disks). We note that this method of running our framework with different scenarios can potentially be used by a DaaS provider as a way of "scoping out" the impact of varying SKUs when making a purchasing decision.

TABLE 2
Experimental Parameters for Evaluating Various Scenarios

Scenario	SLO set	Tenant Ratio	diskC Amortized Cost
SC1	S_1={10tps, 1tps}	20:80	$111
SC2	S_1={10tps, 1tps}	50:50	$111
SC3	S_1={10tps, 1tps}	80:20	$111
SC4	S_1={10tps, 1tps}	20:80	$88
SC5	S_1={10tps, 1tps}	50:50	$88
SC6	S_1={10tps, 1tps}	80:20	$88
SC7	S_2={100tps, 1tps}	20:80	$111
SC8	S_2={100tps, 1tps}	50:50	$111
SC9	S_2={100tps, 1tps}	80:20	$111
SC10	S_2={100tps, 1tps}	20:80	$88
SC11	S_2={100tps, 1tps}	50:50	$88
SC12	S_2={100tps, 1tps}	80:20	$88
SC13	S_3={100tps, 10tps}	20:80	$111
SC14	S_3={100tps, 10tps}	50:50	$111
SC15	S_3={100tps, 10tps}	80:20	$111
SC16	S_3={100tps, 10tps}	20:80	$88
SC17	S_3={100tps, 10tps}	50:50	$88
SC18	S_3={100tps, 10tps}	80:20	$88

Tenant ratios divide 10,000 tenants across two SLOs for each scenario. The ssdC SKU amortized cost over 36 months is $125.

3.1 Solutions Using The Framework

Hardware provisioning and tenant scheduling policies are depicted using bubble plots in a 2-dimensional space. Each bubble represents a single hardware SKU with a specific tenant schedule as determined by the coordinates of the center of the bubble. The size of the bubble denotes the number of servers provisioned from that SKU (i.e., α_i in Problem Definition 1). The position of the bubble corresponds to the the tenant scheduling policy represented by vector $\vec{b_i}$ in the problem definition. That is, the y coordinate is the number of high-performance tenants scheduled on that SKU, and the x coordinate is the number of low-performance tenants. Recall that Definition 4 allows a single hardware SKU to be used multiple ways with different tenant scheduling policies. Thus, even though we have only two types of servers, *ssdC* and *diskC*, a single plot may contain more than two bubbles.

Next, we discuss the hardware provisioning and tenant scheduling policies obtained for each set of SLOs in turn.

3.1.1 SLO Set 1 – 10tps and 1tps

As shown in Figs 4(a),(b), this set of SLOs results in linear SKU performance characterizing functions for both the *ssdC* and *diskC* SKUs. Since the *ssdC* SKU can serve twice the number of 10tps *and* 1tps tenants as the *diskC* SKU, we expect the optimal hardware provisioning policy to favor *ssdC* servers, given that the price of an *ssdC* server is less than twice the price of a *diskC* server.

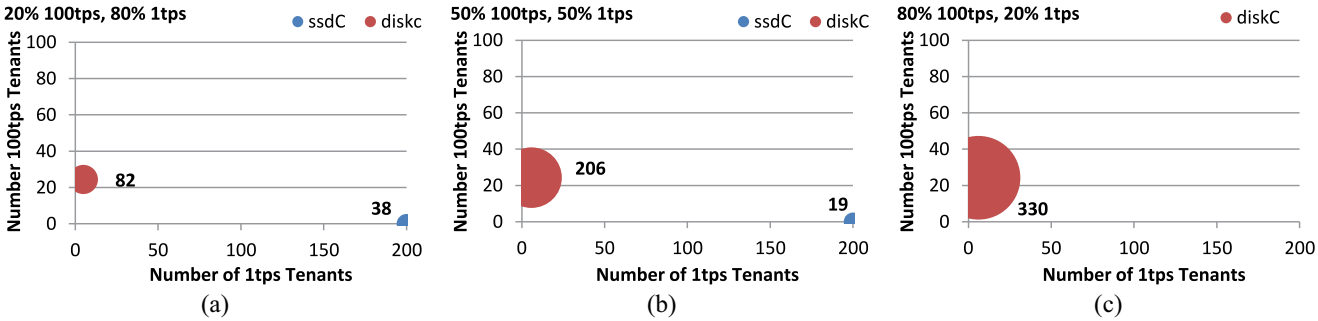

Fig. 9. Solutions for (a) SC7 - $13,861; (b) SC8 - $25,264; (c) SC9 - $36,667 (see Table 2 for details). Circle positions indicate tenant scheduling policy and circle size/annotation indicate hardware SKU provisioning policy.

Fig. 8(a)–(c) shows the optimal solutions for scenarios SC1, SC2 and SC3 (*diskC* costs 10% less than *ssdC*). As expected, the optimal provisioning policy uses only *ssdC* SKUs. Note that since exactly one SKU is used, the ratio of tenants scheduled on each server (determined by the y and the x coordinates) corresponds exactly to the total tenant ratio (20:80, 50:50 and 80:20 in Fig. 8(a), (b) and (c) respectively). The total costs of the optimal solutions in each case are indicated at the bottom of figure. We can see that as the proportion of 10tps tenants increases from (a) to (c), more servers are required, which increases the total solution cost.

Next, we evaluated the scenarios SC4-SC6 (i.e., the *diskC* SKU is 30% cheaper than the *ssdC* SKU). The optimal policies obtained in this case are identical to those of scenarios SC1-SC3 shown in Fig. 8 (and hence the figures are omitted). Since the solutions only used *ssdC* SKUs, the change in the *diskC* SKU cost does not affect the optimal solution cost. Again, this is expected given the much higher performance delivered by the *ssdC* servers for this set of SLOs.

These results suggests that since the *diskC* SKU used in our evaluation delivers roughly half the performance of the *ssdC* SKU, it must cost less than half the *ssdC* SKU to be considered cost-effective. As this experiment (and recent studies [29]) show, the considerable performance benefits obtained by the SSDs may in some cases compensate for the price premium.

3.1.2 SLO Set 2 – 100tps and 1tps

Fig. 9(a)–(c) show the optimal hardware provisioning and the tenant scheduling policies for scenarios SC7, SC8, and SC9 respectively (*diskC* costs 10% less than *ssdC*). As expected, the cheaper *diskC* SKU plays a large role in the optimal solution. In fact, when the tenant mix contains a large proportion of 100tps tenants (Fig. 9(c)), the *ssdC* SKU is not used at all! Furthermore, note that even when the *ssdC* servers are used (Fig. 9(a) and (b)), only the 1tps tenants are scheduled on these servers. These results are somewhat counter-intuitive, since the high-end SKU is scheduled only with the low-end tenants.

In Fig. 10(a)–(c), we show the optimal solutions for scenarios SC10, SC11 and SC12 (*diskC* costs 30% less than *ssdC*). Now, compared to the results shown in Fig. 9, we observe that the hardware provisioning policy uses even fewer *ssdC* servers due to their higher relative cost.

An interesting observation from these results is that in the recommended hardware provisioning policy, the ratio of the number of servers of one SKU over the number of servers of the other SKU is very large. Examples of this can be found for SC8 and SC11, Figs. 9(b), 10(b) respectively, where the number of *ssdC* servers is an order magnitude less than the number of *diskC* servers. An alternative (albeit suboptimal) SKU provisioning strategy is to simply use only *diskC* servers, and ignore *ssdC* altogether (or vice versa). The advantage of this strategy is that it produces a homogeneous cluster that is easier to manage and administer. In Section 3.2, we discuss this and other suboptimal (from the initial hardware provisioning cost perspective) alternatives and their costs.

3.1.3 SLO Set 3 – 100tps and 10tps

Here we consider SLO Set S_3 corresponding to scenarios SC13-15 and SC16-18 in Table 2. Fig. 11 plots SC13-15 where the *diskC* SKU costs 10% less than the *ssdC* SKU, and

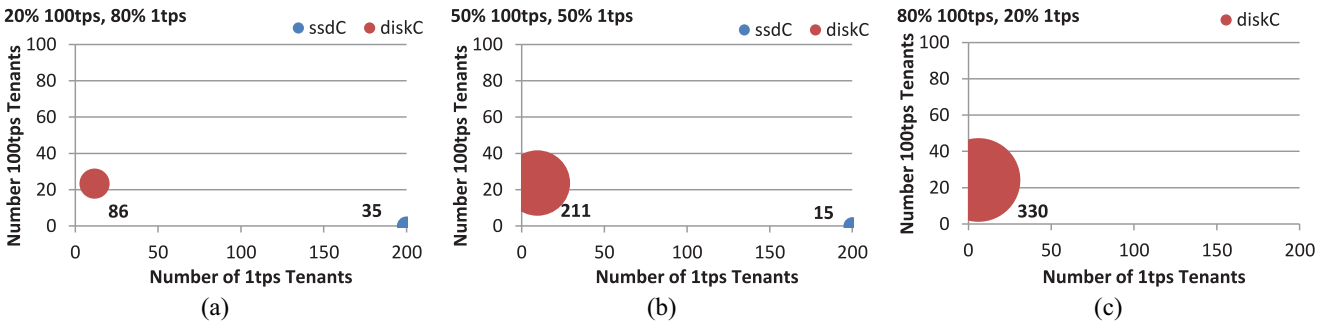

Fig. 10. Solutions for (a) SC10 - $11,900; (b) SC11 - $20,338; (c) SC12 - $28,875 (see Table 2 for details). Circle positions indicate tenant scheduling policy and circle size/annotation indicate hardware SKU provisioning policy.

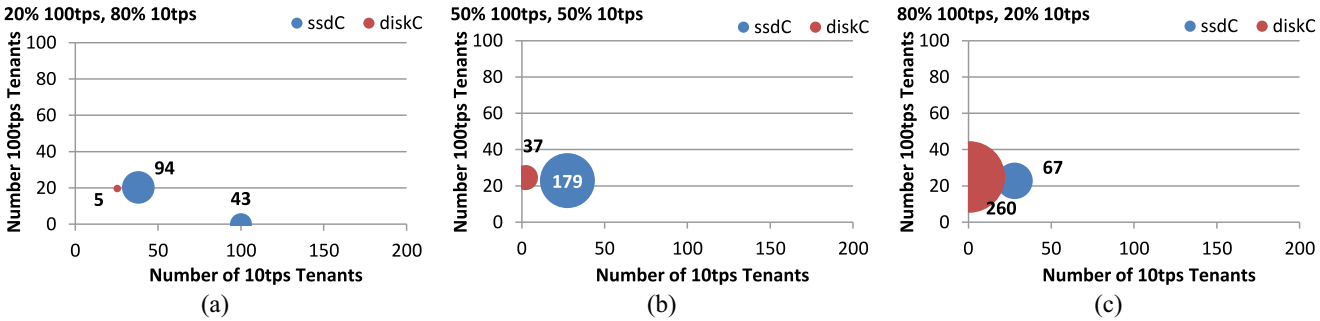

Fig. 11. Solutions for (a) SC13 - $17,681; (b) SC14 - $26,486; (c) SC15 - $37,264 (see Table 2 for details). Circle positions indicate tenant scheduling policy and circle size/annotation indicate hardware SKU provisioning policy.

Fig. 12 plots SC16-18 for the case where the *diskC* SKU costs 30% less.

Interestingly, for this set of SLOs, in some scenarios, the optimal solution uses the *ssdC* SKU with *two* different tenant scheduling policies. As seen in Figs. 11(a), 12(a), there are two blue bubbles representing *ssdC* servers – one bubble represents servers that are scheduled with only 10tps tenants and the other represents servers that are scheduled with a mix of tenants.

Since we have a 100tps SLO in S_3, the *diskC* servers provide better value because they can handle the same number of 100tps tenants at a lower price. This is why we predominantly see *diskC* servers in the solutions as the tenant ratio shifts toward the high-performance tenants. Similar to Fig. 10, as we decrease the cost of the *diskC* SKU (Fig. 12), or increase the number of 100tps tenants (SC15 in Fig. 11 and SC18 in Fig. 12), the optimal solution provisions mostly cheaper *diskC* servers.

Note that in Fig. 11(c), the *diskC* servers (red bubble) are scheduled with just one 10tps tenant per server. A simpler solution (with a possibly higher cost) might be to simply schedule no 10tps tenants on the *diskC* servers. Such solutions are discussed in the following section.

3.2 Suboptimal Solutions – Simplicity vs Cost

In this section, we discuss issues related to the simplicity and manageability of the hardware provisioning and tenant scheduling policies dictated by our framework. At the outset, note that our notion of "total cost" is simplistic as it is only defined in terms of the costs of individual servers. In cloud deployments, issues such as cluster manageability also carry a cost and play an important role in provisioning decisions. In particular, heterogeneous clusters comprised of multiple SKUs can be harder to maintain, manage, and administer compared to homogeneous clusters comprised of a single SKU. A related issue is the complexity of scheduling policies. A straightforward scheduling policy (e.g., assign all tenants with SLO s_1 on SKU 1, s_2 on SKU 2, etc.) may simplify hardware provisioning decisions as well as tenant pricing policies. For instance, if tenants of a given SLO class are tied to a certain SKU, then they can be charged at a rate determined by the price of that SKU. In this paper, we do not attempt to quantify the notion of cluster "complexity", but leave that as part of future work. Nevertheless, the additional server costs imposed by simpler hardware provisioning and tenant scheduling policies can be determined.

Table 3 lists four alternative methods to our optimizing framework. In method *ssdC-only*, we use a homogeneous cluster comprised only of the *ssdC* SKU. Note that this method allows a heterogeneous mix of tenants with different SLOs on a server and also allows for different tenant scheduling policies on different *ssdC* servers. Method *diskC-only* is similar, but with *diskC* servers taking the place of the *ssdC* servers. In method *ssdC-hightps*, all of the high-end tenants are scheduled on the *ssdC* servers, and all of the low-end tenants on the *diskC* servers. In method *ssdC-lowtps*, this assignment is reversed. Thus, in the latter two policies, the SLOs are tied to SKUs. Note that another possible method is to provision a homogeneous cluster *and* maintain a homogeneous tenant scheduling policy each server. We omit this method since it is subsumed by the *ssdC-only* and the *diskC-only* methods that allow for both homogeneous and heterogeneous tenant scheduling policies.

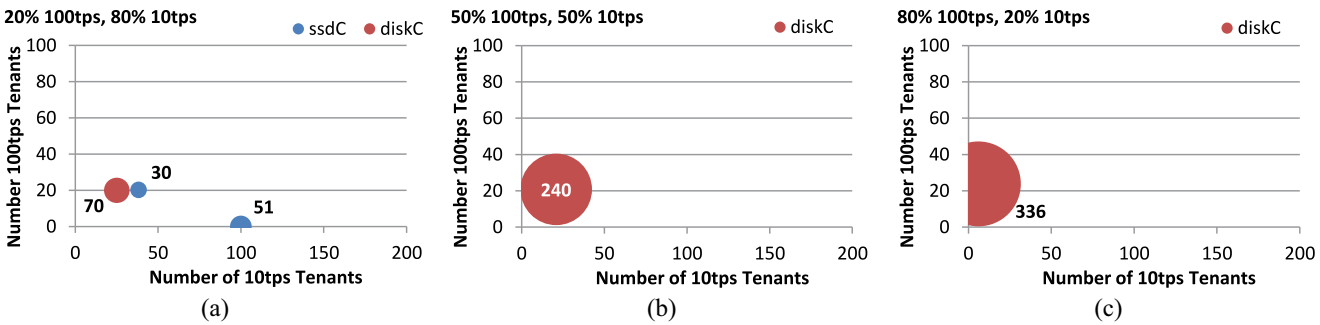

Fig. 12. Solutions for (a) SC16 - $16,250; (b) SC17 - $21,000; (c) SC18 - $29,400 (see Table 2 for details). Circle positions indicate tenant scheduling policy and circle size/annotation indicate hardware SKU provisioning policy.

TABLE 3
Comparing Tenant Scheduling on Two Hardware SKUs

Methods	ssdC SKU	diskC SKU
Optimal	heterogeneous SLOs	heterogeneous SLOs
ssdC-only	heterogeneous SLOs	–
diskC-only	–	heterogeneous SLOs
ssdC-hightps	homogeneous high-perf	homogeneous low-perf
ssdC-lowtps	homogeneous low-perf	homogeneous high-perf

In Figs. 13–15, we plot the total costs obtained by the five methods outlined in Table 3 for the 18 scenarios described Table 2. All solutions are plotted relative to the cost-optimal solution (shown as the left-most bar) discussed in Section 3.1. At a high-level, while in each case there are some solutions that are identical or very close to the optimal solution, *there is no single method that consistently gives a solution that is close to the optimal solution in all scenarios*. For example, while *ssdC-lowtps* seems to match optimal cost in the $S = \{100tps, 1tps\}$ cases, this is not the trend when $S = \{10tps, 1tps\}$. In another case, while the *ssdC-only* solution is optimal for the scenarios depicted in Fig. 13, it is not optimal for the scenarios shown in Fig. 14.

Let us examine a few solutions in more detail. In Fig. 13(a) and (b), which correspond to the $\{10tps, 1tps\}$ SLO scenarios SC1-3 (*diskC* SKU cost 10% less than the *ssdC* SKU) and SC4-6 (*diskC* SKU cost 30% less than the *ssdC* SKU) in Table 2 respectively, the *diskC-only* solution is significantly worse than the optimal solution while the *ssdC-only* solution is optimal. The *ssdC-hightps* method appears increasingly attractive as the proportion of 10tps tenants (high-perf. tenants) grows. The costliest solutions are the *diskC-only* and *ssdC-lowtps* methods. In Fig. 14 (the $S = \{100tps, 1tps\}$ case), the *ssdC-only* and the *ssdC-hightps* methods are expensive solutions in

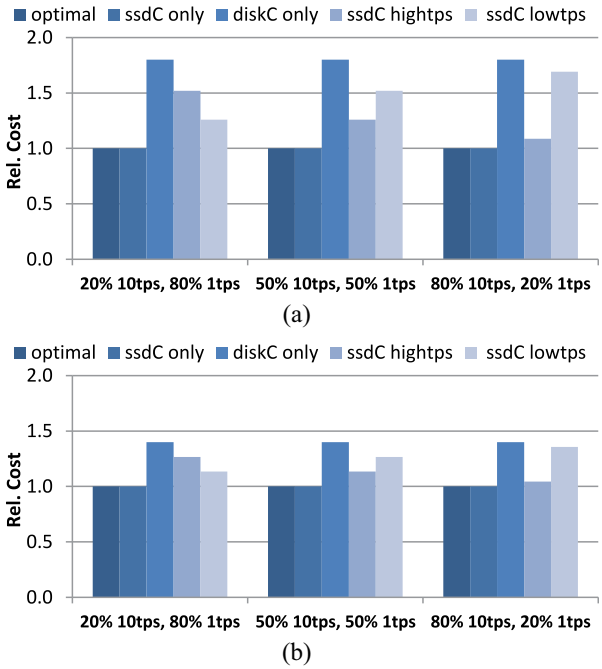

Fig. 13. Relative costs corresponding to solutions for $\{10tps, 1tps\}$ scenarios: (a) SC1-3; (b) SC4-6 (see Table 2) using our framework and 4 simple methods (see Table 3).

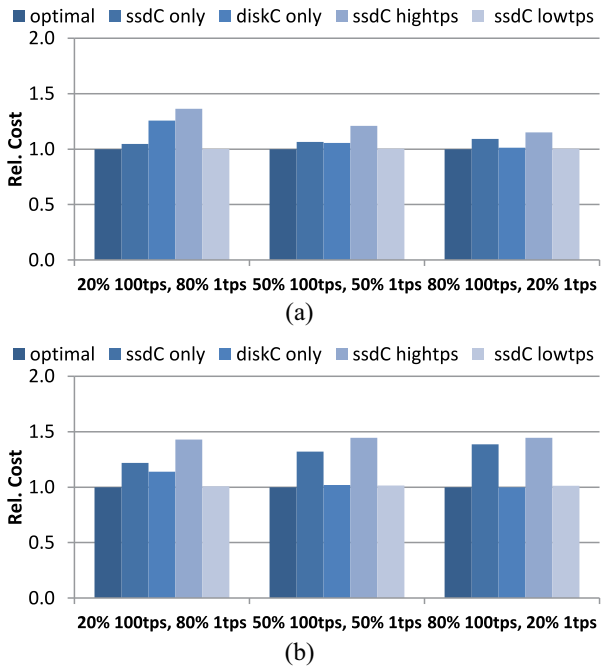

Fig. 14. Relative costs corresponding to solutions for $\{100tps, 1tps\}$ scenarios: (a) SC7-9. (b) SC10-12 (see Table 2) using our framework and 4 simple methods (see Table 3).

Fig. 14(b) and (a) respectively, since the *ssdC* and the *diskC* SKUs can both handle only 25 100 tps tenants, but the *ssdC* server is more expensive. Also, a homogeneous *diskC* cluster is generally more expensive when the tenants skew towards the 1tps SLO. This is because the ssdC SKU can schedule many per 1tps tenants than diskC SKU (Fig. 3). The trends shown in Fig. 15 (for $S = \{100tps, 10tps\}$) are similar to those of Fig. 14 for the same reason.

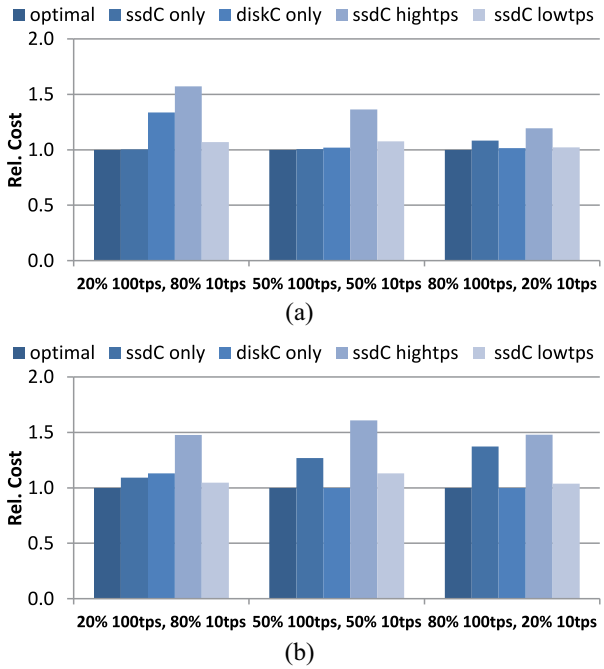

Fig. 15. Relative costs corresponding to solutions for $\{100tps, 10tps\}$ scenarios (a) SC13-15. (b) SC16-18 (see Table 2) using our framework and 4 simple methods (see Table 3).

This analysis shows that simpler provisioning methods may come close to the optimal solution provided by our framework, but no single method produces consistently good solutions. Moreover, these simpler heuristics *still require SKU performance characterization* in order to schedule tenants while adhering to tenant SLOs. Our framework produces low-cost hardware provisioning and tenant scheduling policies for multi-tenant database clusters that are up to 33% less costly than simpler provisioning methods. Thus, the cost benefit of an optimal solution over a suboptimal solution must be weighed against cluster manageability and simplicity.

4 The Cost of Dynamic Environments

Throughout our previous analysis and discussion, we assumed that the DaaS provider only needs to optimize once for a static number of tenants. In this section, we will relax this assumption and consider what happens when the tenant population changes in size and/or SLO make-up (the ratio of high-performance tenants to low-performance tenants).

Specifically, we consider the following problem: first, the DaaS provider has an active set of tenants, T_1, that have been optimally scheduled to a set of provisioned servers according to our optimization framework. In addition, the provider has another set of tenants, T_2, that potentially has a different ratio of high-performance tenants to low-performance tenants. The set T_2 is intermittently active and is also optimally scheduled to provisioned servers using our framework. The problem we consider is: Given that the provider has monitored and profiled the T_2 tenant set, and knows that they will soon become active for time length w, should the provider (i) keep the two tenant populations independently **locally-optimized**; or, (ii) **globally-optimize** all $T_1 \cup T_2$ tenants? (Note that if $w = \infty$, this is the case where the T_2 tenants will remain active indefinitely.)

When faced with these two options, we must account for the cost of the global optimization over different sets of tenants. When considering a global optimization across $T_1 \cup T_2$, we must consider what happens when a T_2 tenant that was scheduled on a particular server is re-scheduled to a different server. Here, we must migrate the tenant's database from one server to another.[5] Since tenant migration consumes resources, this action is not free and so we associate a cost function $g(dest, source)$, in dollars, with migrating a tenant from server *source* to server *dest*. (For simplicity, we assume fixed sized tenants.)

Formally, we define an extension to Problem Statement 1 that compares the cost of keeping tenant populations locally-optimized or globally-optimizing all the tenants. Recall that the solution to Problem Statement 1 is a pair of variables (a, B) where a represents the number of machines of each hardware SKU to buy and B represents the scheduling policy of tenants onto each of the hardware SKUs.

Problem Definition 2. *We are given SLO set S, server SKU multiset M, and two tenant sets T_1 and T_2. Using our framework to solve Problem Statement 1 independently for T_1 and T_2, we get the* ***locally-optimized*** *solutions (a_1, B_1) and (a_2, B_2) respectively. Over a time length w, the amortized server cost of the T_1 solution (a_1, B_1) will be C_1^w and the amortized server cost of the T_2 solution (a_2, B_2) will be C_2^w. Alternatively, the provider can* ***globally-optimize*** *the combined tenant set $T_1 \cup T_2$ using our framework and get a scheduling and provisioning solution $(a_{1\cup2}, B_{1\cup2})$ with a cost of $C_{1\cup2}^w$. However, execution this solution requires migrating the set of tenants $\pi \subseteq T_1 \cup T_2$. This migration incurs the cost $\Sigma_{\forall \tau_i \in \pi} g(to(\tau_i), from(\tau_i))$, where $from(\tau_i)$ and $to(\tau_i)$ provide the source and destination servers for migrating τ_i respectively. The provider should only globally-optimize all $T_1 \cup T_2$ tenants for the upcoming w timespan if:*

$$C_1^w + C_2^w > C_{1\cup2}^w + \Sigma_{\forall \tau_i \in \pi} g(to(\tau_i), from(\tau_i)) \quad (1)$$

In the following section, we first analyze the difference between the sum of the costs of the **locally-optimized** solutions, $C_1^\infty + C_2^\infty$, and the cost of the **globally-optimized** solution, $C_{1\cup2}^\infty$. Then, we consider the cost of migration when globally optimizing the entire tenant population.

4.1 The Cost of Locally-optimal and Globally-optimal solutions

From Equation 1, it is intuitive to see that $C_1^\infty + C_2^\infty - C_{1\cup2}^\infty$ bounds the migration cost if the provider wishes to globally optimize $T_1 \cup T_2$. If the cost difference between the locally-optimal solution and the globally-optimal solution is small, then it means that the cost per tenant migration must be very low to make global re-optimization viable. We now present some results that show how big the cost differences can be.

In Fig. 16, we plot $(C_1^\infty + C_2^\infty)/C_{1\cup2}^\infty$ for various sets of T_1 and T_2 tenants where $S = \{H, L\}$, and H corresponds to the 100tps SLO and L corresponds to the 10tps SLO. In all of the sub-figures, we considered a variety of T_2 populations from 1,000 to 20,000 (x-axis) and the new ratios that we considered are 1H:6L, 1H:4L, 1H:2L, 1H:1L, 2H:1L, 4H:1L, and 6H:1L (the various data series). The three sub-figures (a)–(c) correspond to 10,000 T_1 tenants with tenant ratios 1H:4L, 1H:1L, and 4H:1L respectively.

In Fig. 16(a), we see that if the T_2 population has a SLO ratio skewed toward the 100tps (i.e., 'H') objective, the global re-optimization solution is significantly cheaper (over 5%). Since the T_1 tenants are skewed in a 1H:4L ratio, the 2H:1L, 4H:1L, and 6H:1L are oppositely skewed. Intuitively, separately optimizing for two oppositely skewed populations should result in a higher cost than a solution that has optimized for the combined populations. Furthermore, we notice that as the T_2 population increases in size, dwarfing T_1, the cost differences begin to shrink since a dominating percentage of $(T_1 \cup T_2)$ is T_2, which was optimally scheduled with an optimal server provisioning. We also notice that if the set T_2 has a similar tenant ratio (1H:2L, 1H:4L, 1H:6L – the dashed curves in

5. In environments with high availability (HA) SLAs, tenant databases are replicated across multiple servers and we may only need to re-direct user requests to a replica. This "swap" approach is generally not as expensive as migrating tenant data. However, considering HA SLAs is beyond the scope of this paper and is the focus of future work. For our discussion, we will simply consider a single replication environment, or one with a fixed primary copy, which in turn requires tenant data migration.

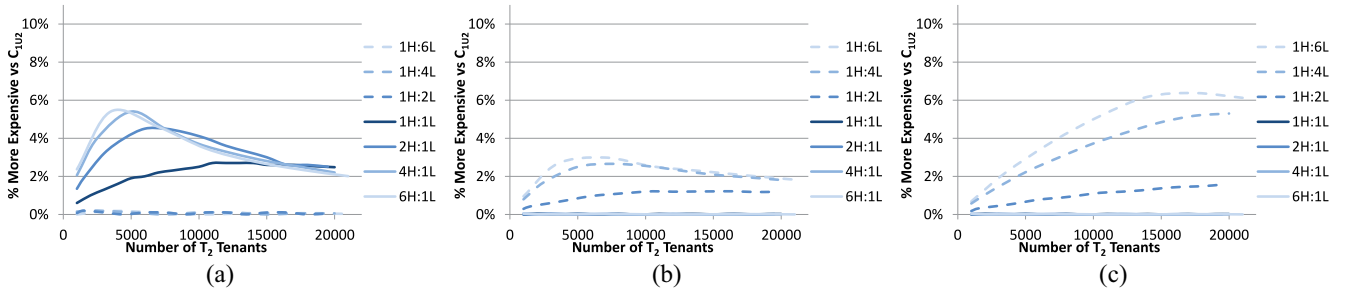

Fig. 16. Relative cost difference between the two **locally-optimized** solutions, $(C_1 + C_2)$, and a single **globally-optimized** solution, $C_{1\cup2}$. The tenants have either "H" (100tps) or "L" (10tps) SLOs. There are 10,000 T_1 tenants. We varied the size of the T_2 tenant set and also the ratio of "H" and "L" tenants in each sub-figure. The diskC server SKU is 10% cheaper than the ssdC server SKU: (a) T1 tenants with 1H:4L SLO ratio. (b) T1 tenants with 1H:1L SLO ratio. (c) T1 tenants with 4H:1L SLO ratio.

Fig. 16(a)) as T_1, then there is a negligible cost difference between the locally-optimized and the globally-optimized solutions.

So from the results shown in Fig. 16(a), we observe that for dealing with Problem Definition 2, the globally-optimized solution is preferred when the T_1 and T_2 tenant sets have opposing SLO ratios. For example, in Fig. 16(a), when T_1 has a 1H:4L SLO ratio and T_2 has a 4H:1L SLO ratio, optimizing the tenant sets separately is very costly.

This observation is also supported by the results shown in Fig. 16(c), where the T_1 population has an SLO ratio of 4H:1L. The analysis shows that the 1H:2L, 1H:4L, and 1H:6L T_2 population curves have higher cost (over 6%). If T_2 has an H-oriented skew (the solid curves), we see that there is a negligible cost difference between the two solutions. In Fig. 16(b), where the T_1 population is balanced at 1H:1L, we see the cost differences when the T_2 tenants are skewed toward the 10tps (L) SLO. However, in this case, the cost difference between the local and the globally-optimized solutions is never over 3%.

4.2 Bounding the Cost of Migration for Global-optimization

Now that we have seen that there can be significant cost differences between having two locally-optimized solutions versus one globally-optimized solution, we can analyze the bounds on the migration costs.

Recall Equation 1 (the cost comparison model from Problem Statement 2): $C_1^w + C_2^w > C_{1\cup2}^w + \Sigma_{\forall\tau_i\in\pi} g(to(\tau_i), from(\tau_i))$, where C_i^w is the cost of the optimized solution for tenant set T_i over the timespan w, and π is the set of tenants that is migrated as a result of global-optimization. For this discussion, let us simplify our model so that all the tenant migrations cost the same amount γ, i.e. $g(to(\tau_i), from(\tau_i)) = \gamma, \forall\tau_i \in \pi$. Therefore, we can simplify our migration cost in Equation 1 to the number of migrations, $|\pi|$, multiplied by the migration cost, γ, and bound the re-optimization migration cost as follows:

$$(C_1^w + C_2^w - C_{1\cup2}^w > |\pi|\gamma) \tag{2}$$

We can apply this equation to the results in Fig. 16(a)–(c). First, let us consider Fig. 16(a), which corresponds to T_1 tenants from the scenario SC13 (10,000 tenants, 1H:4L, see Table 2). Now, consider the case when $|T_2| = 5000$ with a ratio of 4H:1L (in Fig. 16(a)). In this case, the locally-optimized solutions are around 5% more expensive than the globally-optimized solution. If we consider $w = \infty$, then $C_1^w + C_2^w - C_{1\cup2}^w = 66528$.

Now, to consider migration, we have to construct the set π. To compute the set π, we first identified server SKUs from the T_1 solution where more tenants of a given SLO class were scheduled onto the SKU than the $T_{1\cup2}$ solution. Then, we count the tenants that need to be migrated away from this server SKU so that its tenant scheduling policy matches the globally-optimized solution. For example, if the T_1 solution requires 94 ssdC servers with a (20H, 38L) scheduling policy, while the $T_{1\cup2}$ solution requires 274 ssdC servers with a (22H, 33L) scheduling policy, then we calculate that $94 \times (38L - 33L) = 470L$ tenants must be moved/migrated. We only count the tenants that need to be migrated *away* from the server and ignore tenants that need to migrate *to* a server to avoid double counting. After we count the migrating tenants from the T_1 solution, we do the same for the T_2 solution.[6]

Following the step above, we find that $|\pi| = 8285$, thus γ must be less than \$8.03 (since $66528 > 8285\gamma$). Thus, if the cost of migration is greater than or equal to \$8.03, then it is more expensive to globally-optimize all the tenants. Since \$8.03 is the bounding cost of migrating the 1GB tenant, this suggests that it may be advantageous to re-optimize all the tenants if we are faced with this decision only once (i.e. $w = \infty$).

On the other hand, consider the case when T_2 is active for 12hrs and then inactive for the subsequent 12hrs every day (e.g., a diurnal pattern). Thus, $w = 12hrs$, and $C_1^w + C_2^w - C_{1\cup2}^w = 1.28$. Now, if $|\pi| = 8285$, then $\gamma <$ \$0.0001 for migration to be cost effective ($1.28/8285 = 0.0001$) This means that globally-optimizing every 12hrs is only feasible when migration can be done at a very low cost.

To summarize, in this section, we have discussed how (i) tenant populations that vary in size, and (ii) tenant populations that vary in SLO make-up can change the way we apply our optimization framework. Intuitively, optimizing two independent sets of tenants results in a solution that is more costly than optimizing across the combined set of tenants. A globally optimal solution may be cheaper (than the independent optimized solutions), but to dynamically

6. We acknowledge this approach may not be optimal and is a complex problem in its own right. It is the focus of future work.

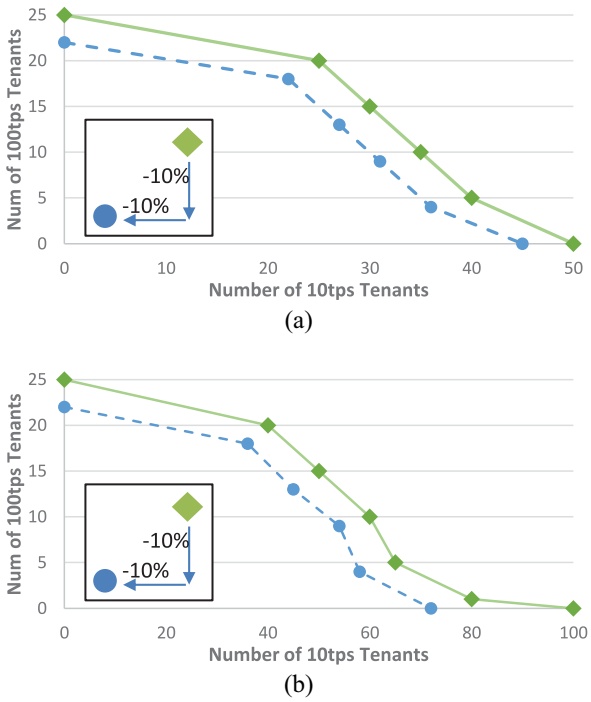

Fig. 17. Buffer-band SKU performance characterizing functions for $S = \{100tps, 1tps\}$ in dashed blue. Original benchmarked functions in green: (a) Performance on the diskC SKU. (b) Performance on the ssdC SKU.

switch to the global optimal solution requires consider tenant migration costs. Using the method described in this section, we can incorporate the migration cost to determine when to switch to the globally-optimal solution.

4.3 The Cost of Tenant SLO Elasticity

Our discussion so far on the dynamicity in the cluster has been limited to changes in the *new subscription ratios* of H tenants to L tenants. In this section we present an analysis of two different approaches in handling tenant SLO elasticity. The scenario we consider is when the L tenants wish to upgrade their SLO from 10tps to 100tps. For instance, such an upgrade may be necessary if a DaaS customer has found that their online e-commerce business has started to take off and they now need a higher transactional throughput.

Let us consider scenario SC13 in Table 2 where we have 2000 100tps tenants and 8000 10tps tenants. An "Aggressive" approach to provisioning the hardware for this scenario is to use Fig. 7 as input into our optimization framework and to produce the solution shown in Fig. 11(a). However, in this case, if some of the 10tps tenants wish to upgrade their SLO to 100tps, the DaaS provider has no choice but to provision new servers and physically move the tenant's data. The DaaS provider cannot upgrade the tenant SLO "in-place" on the server because, as we showed in Fig. 6, the substitution ratio of low performance tenants to high performance tenants is typically r:s where $r > s$. Thus, an Aggressive approach is very sensitive to any perturbations caused by the tenants changing their SLOs, as more servers may need to be provisioned.

On the other hand, consider a "Buffer-band" approach whereby we consciously scale back the characterizing frontier functions that we use in our optimization framework.

TABLE 4
Tenant SLO Upgrade Costs When Using an Aggressive Packing Approach Versus a Buffer-Band Approach (See SC13 Table 2)

Approach	BaseCost	10% Tenant upgrade (provision ssdC)	10% Tenant upgrade (provision diskC)
Aggressive	$17681	$21681	$ 21233
Buffer-band	$21125	$21125	$ 21125

10% of the 10tps tenants upgrade to 100tps. If more servers are needed, we can choose to buy ssdC or diskC SKUs.

In Fig. 17, we show the Buffer-band characterizing function (dashed blue line) where we scale back our benchmark results by 10% in both axes. Feeding the blue characterizing functions into our optimization framework for SC13 results in solutions that are significantly more expensive than the Aggressive approach. In Table 4, we show that the Buffer-band solution is almost 20% more expensive (this is not surprising as our band is scaled back by 10% in two axes). However, if the DaaS provider finds that 10% of the 10tps tenants upgrade their SLO subscriptions to 100tps, the Aggressive approach will require $(8000 \times 10\%)/25 = 32$ (25 is the maximum 100tps limit for both ssdC and diskC) more machines, and this added cost surpasses the cost of the Buffer-band approach. In this specific example, the Buffer-band approach can handle 10% of the 10tps tenants upgrading "in-place" without requiring new servers or data movement. In Table 4, we show two different upgrade costs for the "Aggressive" approach since the provider has the choice of buying either more ssdC or diskC SKUs to accommodate the upgrading tenants. Furthermore, in our analysis here, we have not included data movement costs incurred by the Aggressive approach when upgraded tenants are moved to the new servers.

In the scenario above, if less than 10% of the 10tps tenants are upgraded, then Table 4 suggests that the Aggressive approach is still cheaper (i.e., if the buffer is never fully utilized). Therefore, the efficacy of the Buffer-band approach is tied to the accuracy of the provider's estimates of the tenant elasticity. In general, the cloud provider could use a combination of the Buffer-band approach and provision machines on the fly (the Aggressive method) based not only on their costs but also business decisions, such as how reactive they need to be to upgrade/downgrade requests, and what penalties they are willing to incur to deal with (temporary) SLO violations.

5 Discussion

While the focus of this paper is on performance SLOs in a DaaS, we have not discussed the impact of tenant replication (a common mechanism that is used to support uptime SLAs) on our performance models. While data replication may improve performance for read-mostly workloads, maintaining replica consistency under update-heavy workloads places additional demands on the resources of DaaS providers. A careful study of how to deal with replica consistency and availability while providing performance SLOs is beyond the scope of this paper, but we sketch an initial method to deal with this issue.

For our framework to handle replica updates, we can modify the benchmarking method that is used to determine

the SKU performance characterizing function (Section 2) to account for the extra work that is needed to maintain replica consistency. For example, instead of measuring tenant performance on a single server as we have done, we would measure the tps observed by a tenant whose replicas are placed on r servers and maintained via eager or lazy updates. The functions obtained from such a benchmark can be used as constraints to the optimization problem defined in Section 2.3.

Using our framework, we can pose another interesting question: given a cluster with a specific composition of hardware SKUs, what performance SLOs can the DaaS provider agree to so that it maximizes the number of tenants that can fit on this cluster? For this question, we need to formulate a new objective function that optimizes for $max(|T|)$ in Problem Definition 1 where T is the set of all tenants. The remaining constraints specified in Problem Definition 1 remain the same.

We note that in calculating the amortized monthly costs, we have not accounted for run time energy costs or amortized infrastructure cost. However, these can be accommodated in our framework (provided that there is an accurate model to compute these costs for each SKU) by simply adding these costs to the amortized monthly cost that we use in this paper.

Finally, in this paper we have used an explicit benchmarking-based approach to understand the effects of mixing SLO classes and tenants. However, our framework is modular in that it is possible to leverage other analytic approaches that predict the impact of mixing tenants with different workloads and SLOs [12], [19].

6 Related Work

DBMSs have traditionally been engineered for a single-tenant "on-premises" environment. However, emerging trends indicate that DBMS workloads are moving towards the cloud. In recent literature [1], [3], [32], several systems for providing databases in the cloud have been proposed and discussed.

In [10], issues such as performance, scalability, security, availability and maintenance must be reconsidered in a multi-tenant cloud environment. Furthermore, as shown in [21], cloud infrastructure is a costly investment for DaaS providers. Thus, an important goal in such an environment is to maximize server utilization [13], [15], [29], [33].

As outlined in Section 2.1.3, there are several methods to consolidate multiple tenants on a single server [5], [6], [9], [16], [18], [33], [38]. In particular, methods based on the use of Virtual Machines (VMs) have been studied in [2]. However, the performance overhead caused by VMs (paging [23], contention [31], OS redundancy [18]) may be too expensive for the more data-intensive workloads considered in this paper. Thus, a number of frameworks for building native multi-tenant applications have also been proposed [7], [14], [35].

The first step in providing performance-based SLOs for customers is to model system performance under a realistic multi-tenant workload (Section 2.2). To this end, recent work has focused on formulating and evaluating performance benchmarks in a cloud environment [17], [25], [36]. Complicating factors such as unpredictable load spikes [11], interference between tenants [19], [27] have also been analyzed. Load balancing may require tenant migration [20] or alternatively, reassignment of a tenant's "master" replica. Other work has studied how to benchmark production systems and train performance and resource utilization models without breaking performance SLOs [8], [12]. This paper is different from these prior complementary works because the focus is on developing a framework *for using* SKU performance characterizing models to come up with cost-effective hardware provisioning policies *and* tenant scheduling policies for various performance SLOs. However, such complementary efforts will help formulate a more rigorous and realistic definition for performance SLAs.

SLAs for cloud-based services are usually formulated in terms of uptime/availability guarantees [4]. Other work in this field has considered allowing tenants to choose between SLAs that guarantee different levels of consistency [26] and response times in in-memory column databases [34]. It is likely that SLAs published by DaaS providers in the future will involve a combination of several factors to satisfy customer requirements.

7 Conclusion and Future Work

This paper presents an extended study of the cost-optimization framework for multi-tenant performance SLOs in a DaaS environment first presented in [30]. Our framework requires as input, a set of performance SLOs and the number of tenants in each of these SLOs classes, along with the server hardware SKUs that are available to the DaaS provider. With these inputs, we produce server characterizing models that can be used to provide constraints into an optimization module. By solving this optimization problem, the framework provides a hardware provisioning policy as well as a tenant scheduling policy for the selected server SKUs. We have evaluated our framework, and shown that in many cases a mixed hardware cluster is optimal. We have also explored the impact of simpler hardware provisioning and tenant scheduling policies. In addition, we have also shown how our framework can be extended to deal with dynamic changes in the workload mix and tenant elasticity.

To limit the scope of our study, we have made some simplifying assumptions on aspects such as performance metrics, tenant workload, and multi-tenancy control mechanism. Relaxing these assumptions provides a rich direction for future work. One direction for future work is to include the impact of replication and load-balancing in our framework, perhaps building on the ideas presented in [28]. Additionally, while our experimental evaluation uses average performance as an SLO metric, it could be extended to include variance as well (as implied by the use of random variables in Definition 2). Imbalanced load or flash-crowd effects could be modeled in our framework as additional tenant classes with high performance requirements – this would produce a hardware "over-provisioning" policy to deal with these effects. If workload spikes are detected in practice, tenants could be dynamically re-scheduled on these extra machines to maintain performance objectives. In addition, while the tenant classes used in this paper have

different memory and disk requirements, other workloads should be considered as well. Finally, in our framework we have taken an approach of explicitly benchmarking the tenant workload classes and mixes, but our framework could be extended to take a more analytical approach that predicts the impact on performance of different workload mixes, perhaps by using multi-query optimization-based approach to estimate the impact on performance [12], [19]. Optimization of tenant and hardware profiling is another challenging and important direction of future work.

Acknowledgments

The authors would like to thank David DeWitt, Alan Halverson, and Eric Robinson for valuable discussions and feedback on this project. This research was supported in part by a grant from the Microsoft Jim Gray Systems Lab, and in part by the US National Science Foundation under grant IIS-0963993.

References

[1] D. J. Abadi, "Data management in the cloud: Limitations and opportunities," *IEEE Data Eng. Bull.*, vol. 32, no. 1, pp. 3–12, Mar. 2009.
[2] A. Aboulnaga, K. Salem, A. A. Soror, U. F. Minhas, P. Kokosielis, and S. Kamath, "Deploying database appliances in the cloud," *IEEE Data Eng. Bull.*, vol. 32, no. 1, pp. 13–20, Mar. 2009.
[3] D. Agrawal, A. E. Abbadi, S. Antony, and S. Das, "Data management challenges in cloud computing infrastructures," in *Proc. Int. Conf. 6th DNIS*, Aizuwakamatsu, Japan, 2010.
[4] *Amazon* [Online]. Available: http://aws.amazon.com/ec2-sla/
[5] S. Aulbach, T. Grust, D. Jacobs, A. Kemper, and J. Rittinger, "Multi-Tenant databases for software as a service: Schema-mapping techniques," in *Proc. SIGMOD*, Vancouver, BC, Canada, 2008.
[6] S. Aulbach, D. Jacobs, A. Kemper, and M. Seibold, "A comparison of flexible schemas for software as a service," in *Proc. SIGMOD*, Providence, RI, USA, 2009.
[7] S. Aulbach, M. Seibold, D. Jacobs, and A. Kemper, "Extensibility and data sharing in evolving multi-tenant databases," in *Proc. IEEE 27th ICDE*, Hannover, Germany, 2011.
[8] S. Babu, N. Borisov, S. Duan, H. Herodotou, and V. Thummala, "Automated experiment-driven management of database systems," in *Proc. 12th Conf. HotOS*, Berkeley, CA, USA, 2009.
[9] P. Bernstein *et al.*, "Adapting microsoft SQL server for cloud computing," in *Proc. IEEE 27th ICDE*, Hannover, Germany, 2011.
[10] C.-P. Bezemer and A. Zaidman, "Multi-tenant SaaS applications: Maintenance dream or nightmare?" in *Proc. IWPSE-EVOL*, Antwerp, Belgium, 2010.
[11] P. Bodik, A. Fox, M. J. Franklin, M. I. Jordan, and D. A. Patterson, "Characterizing, modeling, and generating workload spikes for stateful services," in *Proc. 1st SoCC*, Indianapolis, IN, USA, 2010.
[12] P. Bodik, R. Griffith, C. Sutton, A. Fox, M. I. Jordan, and D. A. Patterson, "Automatic exploration of datacenter performance regimes," in *Proc. ACDC*, Barcelona, Spain, 2009.
[13] H. Cai, B. Reinwald, N. Wang, and C. J. Guo, "SaaS multi-tenancy: Framework, technology, and case study," *Int. J. Cloud Applicat. Comput.*, vol. 1, no. 1, pp. 62–77, 2011.
[14] Y. Cao *et al.*, "ES2: A cloud data storage system for supporting both OLTP and OLAP," in *Proc. IEEE 27th ICDE*, Hannover, Germany, 2011.
[15] J. S. Chase, D. C. Anderson, P. N. Thakar, and A. M. Vahdat, "Managing energy and server resources in hosting centers," in *Proc. 18th SOSP*, New York, NY, USA, 2001.
[16] F. Chong, G. Carraro, and R. Wolter. (2006). *Multi-tenant Data Architecture* [Online]. Available: http://msdn.microsoft.com/en-us/library/aa749086.aspx
[17] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears, "Benchmarking cloud serving systems with YCSB," in *Proc. 1st SoCC*, Indianapolis, IN, USA, 2010.
[18] C. Curino *et al.*, "Relational cloud: A database-as-a-service for the cloud," in *Proc. CIDR*, Pacific Grove, CA, USA, 2011.
[19] J. Duggan, U. Cetintemel, O. Papaemmanouil, and E. Upfal, "Performance prediction for concurrent database workloads," in *Proc. SIGMOD*, Athens, Greece, 2011.
[20] A. J. Elmore, S. Das, D. Agrawal, and A. E. Abbadi, "Zephyr: Live migration in shared nothing databases for elastic cloud platforms," in *Proc. SIGMOD*, Athens, Greece, 2011.
[21] J. Hamilton, "Cooperative expendable micro-slice servers (CEMS): Low cost, low power servers for internet-scale services," in *Proc. 4th Biennial CIDR*, Pacific Grove, CA, USA, 2009.
[22] D. Hastorun *et al.*, "Dynamo: Amazons highly available key-value store," in *Proc. 21st SOSP*, Washington, DC, USA, 2007.
[23] G. Hoang, C. Bae, J. Lange, L. Zhang, P. Dinda, and R. Joseph, "A case for alternative nested paging models for virtualized systems," *Comput. Archit. Lett.*, vol. 9, no. 1, pp. 17–20, Jan. 2010.
[24] E. P. C. Jones, D. J. Abadi, and S. Madden, "Low overhead concurrency control for partitioned main memory databases," in *Proc. SIGMOD*, Indianapolis, IN, USA, 2010.
[25] D. Kossmann, T. Kraska, and S. Loesing, "An evaluation of alternative architectures for transaction processing in the cloud," in *Proc. SIGMOD*, Indianapolis, IN, USA, 2010.
[26] T. Kraska, M. Hentschel, G. Alonso, and D. Kossmann, "Consistency rationing in the cloud: Pay only when it matters," in *Proc. VLDB*, Lyon, France, 2009.
[27] T. Kwok and A. Mohindra, "Resource calculations with constraints, and placement of tenants and instances for multi-tenant SaaS applications," in *Proc. 6th ICSOC*, Sydney, NSW, Australia, 2008.
[28] W. Lang, J. M. Patel, and J. F. Naughton, "On energy management, load balancing and replication," in *Proc. SIGMOD Rec.*, New York, NY, USA, 2009.
[29] W. Lang, J. M. Patel, and S. Shankar, "Wimpy node clusters: What about non-wimpy workloads?" in *Proc. 6th Int. Workshop DaMoN*, Indianapolis, IN, USA, 2010.
[30] W. Lang, S. Shankar, J. M. Patel, and A. Kalhan, "Toward multi-tenant performance SLOs," in *Proc. IEEE 28th ICDE*, Washington, DC, USA, 2012.
[31] A. Menon, J. R. Santos, Y. Turner, G. J. Janakiraman, and W. Zwaenepoel, "Diagnosing performance overheads in the xen virtual machine environment," in *Proc. VEE*, Chicago, IL, USA, 2005.
[32] R. Ramakrishnan, B. Cooper, A. Silberstein, and U. Srivastava, "Data serving in the cloud," in *Proc. LADIS*, 2009.
[33] B. Reinwald. (2010). *Multitenancy* [Online]. Available: http://www.cs.washington.edu/mssi/2010/BertholdReinwald.pdf
[34] J. Schaffner, B. Eckart, D. Jacobs, C. Schwarz, H. Plattner, and A. Zeier, "Predicting in-memory database performance for automating cluster management tasks," in *Proc. IEEE 27th ICDE*, Hannover, Germany, 2011.
[35] O. Schiller, B. Schiller, A. Brodt, and B. Mitschang, "Native support of multi-tenancy in RDBMS for software as a service," in *Proc. 14th EDBT*, Uppsala, Sweden, 2011.
[36] P. Shivam, V. Marupadi, J. Chase, T. Subramaniam, and S. Babu, "Cutting corners: Workbench automation for server benchmarking," in *Proc. ATC USENIX*, Berkeley, CA, USA, 2008.
[37] S. Srikantaiah, A. Kansal, and F. Zhao, "Energy-aware consolidation for cloud computing," in *Proc. HotPower*, Berkeley, CA, USA, 2009.
[38] C. D. Weissman and S. Bobrowski, "The design of the force.com multitenant internet application development platform," in *Proc. SIGMOD*, Providence, RI, USA, 2009.
[39] L. Zhou and W. D. Grover, "A theory for setting the "safety margin" on availability guarantees in an SLA," in *Proc. 5th Int. Workshop DRCN*, Ischia, Italy, 2005.

Willis Lang is a Research SDE with Microsoft Corporation at the Jim Gray Systems Lab in Madison, WI. He received the B.Math. degree from the University of Waterloo, the M.Sc. degree from the University of Michigan, and the Ph.D. degree from the University of Wisconsin-Madison. His current research interests include data management, cost-effective computing, and computer systems.

Srinath Shankar works as a Research SDE with Microsoft Corporation at the Jim Gray Systems Lab in Madison, WI, USA. He received the B.Tech. degree from the Indian Institute of Technology, Madras and the M.S. and Ph.D. degrees in computer science from the University of Wisconsin-Madison.

Jignesh M. Patel received the Ph.D. degree from the University of Wisconsin-Madison, where he is currently a Professor in Computer Science. He is an ACM Distinguished Scientist, and also a member of ACM and IEEE. His current research interests include high performance and scalable big data management.

Ajay Kalhan is a Principal Development Lead for SQL Azure at Microsoft Corporation.