This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2319095, IEEE Transactions on Parallel and Distributed Systems

SmartSLA: Cost-sensitive Management of Virtualized Resources for CPU-bound Database Services

Pengcheng Xiong, Yun Chi, Shenghuo Zhu, Hyun Jin Moon, Calton Pu, and Hakan Hacıgümüş

Abstract—Virtualization-based multi-tenant database consolidation is an important technique for database-as-a-service (DBaaS) providers to minimize their total cost which is composed of SLA penalty cost, infrastructure cost and action cost. Due to the bursty and diverse tenant workloads, over-provisioning for the peak or under-provisioning for the off-peak often results in either infrastructure cost or service level agreement (SLA) penalty cost. Moreover, although the process of scaling out database systems will help DBaaS providers satisfy tenants' service level agreement, its indiscriminate use has performance implications or incurs action cost. In this paper, we propose SmartSLA, a cost-sensitive virtualized resource management system for CPU-bound database services which is composed of two modules. The system modeling module uses machine learning techniques to learn a model for predicting the SLA penalty cost for each tenant under different resource allocations. Based on the learned model, the resource allocating module dynamically adjusts the resource allocation by weighing the potential reduction of SLA penalty cost against increase of infrastructure cost and action cost. SmartSLA is evaluated by using the TPC-W and modified YCSB benchmarks with dynamic workload trace and multiple database tenants. The experimental results show that SmartSLA is able to minimize the total cost under time-varying workloads compared to the other cost-insensitive approaches.

Index Terms—Cloud computing, virtualization, database systems, multitenant databases

1 Introduction

Database-as-a-service (DBaaS) is one of the most useful applications of cloud computing [5], [11], as shown in Figure 1. In DBaaS, service providers deliver traditional database functionality (such as data definition, storage and retrieval) on a subscription basis over the Internet. Such services are usually provided through standard APIs (e.g., SOAP or REST) and therefore clients can perform CRUD (create, read, update and delete) operations on their databases by using standard SQL. From the clients' point of view, desirable features of DBaaS include lower upfront investment, pay-as-you-go pricing, and reliable performance as specified by the service level agreements (SLAs).

A key technique from the service providers' perspective is *multi-tenant* databases. In a multi-tenant environment, providers consolidate multiple clients (or tenants) such that the clients can make use of the shared resources while still satisfying their service level agreements [5], [11]. Such consolidation allows operating cost reduction without revenue loss and therefore is extremely attractive to DBaaS providers. Examples of multi-tenant DBaaS include

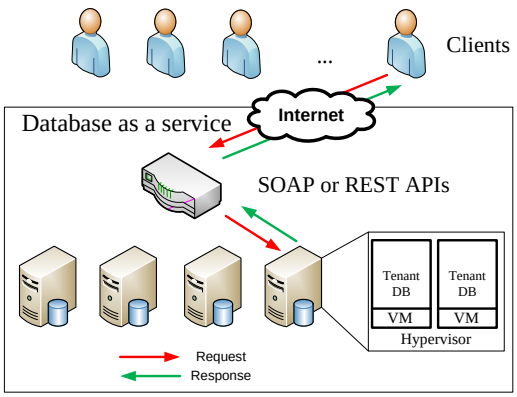


Fig. 1: A shared hardware multi-tenant database architecture

Salesforce.com's Force.com[1], which provides data services in its toolkit for building applications, and Amazon's SimpleDB[2], which provides an API for creating data stores (which can, in turn, be used for applications or pure data storage). Consolidation in multi-tenant databases can take place at different levels. Such levels include *shared hardware*, *shared virtual machine*, *shared DBMS*, and *shared table* [29]. In this paper, we focus on the *shared hardware* model, where each *tenant* database runs in its own virtual machine (VM).[3]

- *P. Xiong, Y. Chi, S. Zhu, and H. Hacıgümüş are with NEC Labs America, Cupertino, CA, 95014. E-mail: {pxiong,ychi,szhu,hakan}@nec-labs.com*
- *C. Pu is with the School of Computing, Georgia Institute of Technology, Atlanta, GA, 30332. E-mail: calton@cc.gatech.edu*
- *H. Moon is with Google Inc, Mountain View, CA, 94043. E-mail: hjlovesshinae@gmail.com*

1. http://www.salesforce.com/
2. http://aws.amazon.com/simpledb/
3. There are two main reasons for us to focus on this level of sharing. First, current virtualization technologies have matured enough to be able to pack a relatively large number of VMs into a single host, thereby increasing the cost efficiency of infrastructure resources [31]. Second, sharing at the level of private virtual machine allows us to leverage virtual machine technologies in order to explicitly control the resource allocation of each individual tenant.

We define the problem as how to help DBaaS providers to achieve their ultimate business goal of minimizing their total cost which is composed of three parts.

(1) SLA penalty cost which is defined as the penalty that a DBaaS provider has to pay due to the violation of service level agreement. For example, given a 1998 world cup workload [4] which contains millions of queries, SLA penalty cost is the total price that a DBaaS provider has to pay for all the queries that violate service level agreement. Because multiple clients share the same server in multitenancy setting, they compete with each other for the same system resources such as CPU cycle, memory, and I/O bandwidth. As a result, without proper resource management, a DBaaS provider may suffer from a high SLA penalty cost.

(2) Infrastructure cost which is defined as the cost that a DBaaS provider has to pay for the infrastructure usage. For example, given a 1998 world cup workload [4] which lasts from 3pm to 10pm, infrastructure cost is the total price that a DBaaS provider has to pay for renting the hardware for 7 hours. DBaaS providers can utilize additional *unlimited* resources, such as additional database slaves, in an on-demand fashion. This latter offering is actually one of the attractive features of the cloud model, namely *pay-as-you-go* with *an illusion of infinite resources*. However, adding additional database replicas involves infrastructure cost (e.g., adding more nodes).

(3) Action cost which is defined as the cost that a DBaaS provider has to pay for the performance implications in the process of scaling out database systems. Database systems are difficult to scale. The process of scaling out database systems is time consuming which involves actions such as data migration and load balancing. Although the process will help DBaaS providers satisfy tenants' service level agreement, its indiscriminate use has performance implications or incurs action cost.

We believe that in order to minimize the total cost, one has to conduct the following two types of analysis.

Local Analysis : The first issue is to identify the right configuration of virtualization resources (e.g., CPU, memory etc.) for a single client to meet the SLA. Answers to such a question are not straightforward as they depend on many factors such as the current workload from the client and the client-specific SLAs (e.g., a gold client may have a different SLA from a silver one).

Global Analysis : The second issue is to allocate resources among multiple clients based on the current system status. For example, how much CPU share or memory should be given to the gold client versus the silver one, when a new database replica should be started, etc. Answers to such decisions obviously rely on the result of the above *Local Analysis* decisions.

In this paper, we minimize DBaaS providers' total cost by conducting these local and global analysis. Our solution is implemented in a real system called SmartSLA[4], which is a cost-sensitive resource management system.

The rest of this paper is organized as follows. We provide background information about SLAs and describe our system setting in Section 2. In Section 3, we describe the system modeling module of SmartSLA. In Section 4, we investigate the performance interference among database VMs sharing the same hypervisor. In Sections 5 we describe the resource allocating module. We evaluate the whole system of SmartSLA in Section 6. We discuss related work in Section 7 before we make conclusions in Section 8.

2 Background

In this section, we provide background information. We first introduce the service level agreement (SLA) model that we use. Then, we give a high-level description of the test bed for our experimental studies.

2.1 Service Level Agreements

Service level agreements (SLAs) are contracts between a service provider and its clients. SLAs may specify or depend on a wide variety of criteria, such as latency, reliability, availability, throughput or security. SLAs are usually in the form of service level objectives (SLOs). One such example is Amazon's key-value storage system Dynamo [13], which guarantees the 99-percentile query latency under a given peak workload. In this paper, we focus on SLAs about SQL query response time. That is, an SLA is a function $F(t)$, where t is the response time of a query and $F(t)$ is the corresponding penalty cost if the SQL query response time is t.

While $F(t)$ may have various shapes, we believe that a step-shaped function is a more natural choice used in the real-world contracts as it is easy to describe in natural language [8]. Figure 2 shows a step-shaped $F(t)$, where the cost to the service provider is $cost$ if the response time of a query q is longer than X_q.

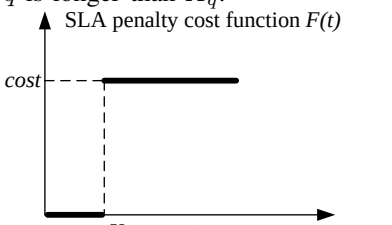


Fig. 2: SLA penalty cost function $F(t)$

For each query q, we denote its start time and the response time by q_{start} and q_{time}, respectively. For the moment we assume $cost = 1$ for ease of discussion (we will generalize this later). Then we have the SLA penalty cost of query q as

$$P(q) = \begin{cases} 0 & \text{if } q_{time} \leq X_q \\ 1 & \text{otherwise} \end{cases}$$

In addition, we define the average SLA penalty cost *AC* as the SLA penalty cost over the total number of queries L, i.e.,

$$AC = \frac{1}{L} \sum_{q} P(q).$$

4. SmartSLA stands for "Resource Management for Resource-Sharing Clients based on Service Level Agreements".

For example, if the client sends 10 queries and 5 of them miss the deadline, then the SLA penalty cost is 5 and the average SLA penalty cost is 0.5.

2.2 Our Test Bed

2.2.1 Architecture

The architecture of our evaluation deployment is shown in Figure 3. Our test bed consists of 10 physical machines. Seven machines are used for hosting database VMs with Xen 3.0 [6]. We use default credit-based CPU scheduler in Xen which is a proportional fair share CPU scheduler. We use "Cap" to limit the maximum amount of CPU that a domain will be able to consume, even if the host system has idle CPU cycles. The cap is expressed in percentage of one physical CPU, e.g., 50% CPU share is half a CPU. Similarly, we leverage memory "ballooning" to hot add and remove memory from a running VM. Two of the machines are used for gold and silver client emulators which issue queries to databases. The last machine runs our SmartSLA. SmartSLA consists of two main components:

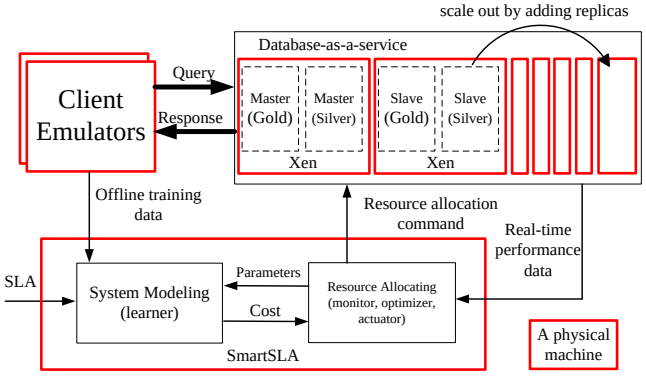


Fig. 3: The architecture of our test bed.

a system modeling module, which mainly answers the *Local Analysis* questions, and a resource allocating module, which mainly answers the *Global Analysis* questions. The system modeling module uses machine learning techniques to learn a model that predicts the potential SLA penalty cost for each client under different resource allocations. Based on the learned model, the resource allocating module dynamically adjusts the system resource allocation in order to minimize the expected total cost to the service provider as shown in Figure 3.

Most virtual machine hypervisor products currently provide mechanisms for controlling the allocating of CPU and memory resources to VMs. However, few products provide mechanisms for effectively controlling other resources, such as I/O bandwidth [31]. Hence, in this study we focus mainly on CPU-bound queries, where CPU and memory are bottleneck resources.

We use MySQL v5.0 with InnoDB storage engine as the database server and use the built-in MySQL replication functionality for scale-out solution. In MySQL, replication enables data from one MySQL database server (the master) to be replicated to one or more MySQL database servers (the slaves). Replication is asynchronous and as a consequence, slaves need not to be always connected to the master in order to receive updates from the master. Each part of the MySQL replication (either master or slave) is hosted in one Xen VM.

We develop workload generators to emulate clients where new queries are generated independently of the completion of previous queries. The arrival rate of queries follows a Poisson distribution with different rate set in each test. We choose Poisson distribution because it is widely used to model independent arrival queries to a database. Extending our work by including non-Poisson arrival is one of our future directions.

In this work we focus on multi-tenant database support for OLTP applications. These applications are characterized by high variance in popularity, small data sizes, and unpredictable and time-varying workload intensities [34], [36]. These characteristics provide guidelines directing our system architecture design and experimental evaluation. For example, multitenant databases in these applications tend to be in the range of hundreds of megabytes to a few gigabytes [34], [36]. We repeat these representative data sizes in our experiments. We have tried two benchmarks, TPC-W [1] and a modified version of Yahoo Cloud Serving Benchmark (YCSB) [10] with different configuration of hosts as shown in Table 1.

TABLE 1: Configuration of hosts

Testbed	TPC-W	YCSB
CPU	AMD 3.0GHz Dual-cores	Intel Xeon 2.40GHz Hexa-core
Memory	2 GB	16 GB
Storage	7200 RPM local disk	7200 RPM local disk

2.2.2 TPC-W

The setting of the benchmark is 100 Emulated Browsers(EB) with 10K items. The whole database size is about 280MB. We use the TPC-W *Ordering* mix for workload mix in which the browsing requests and the ordering requests are 50% each. The requests contain both `write` and `read` queries. The `write` queries are sent only to the master while the `read` queries are sent only to the slaves in a round-robin manner. According to previous work [30], and confirmed by our test, these queries are CPU-bound when the database can reside in the memory.

For each TPC-W query, we assign it an SLA penalty cost function as shown in Figure 2. To decide the X_q value in the SLA, we run each query in a virtual machine with maximum possible resources, i.e., 100% CPU share and 1024MB memory, to get the average response time. Then we set X_q to two times the average response time.

2.2.3 Yahoo Cloud Serving Benchmark

We use a heavily modified version of the Yahoo Cloud Serving Benchmark (YCSB) [10]. While YCSB was originally designed exclusively for key-value stores, we begin with an extended version used in prior work [12]. The whole database size is 1.21GB. The queries contain `read`, `update` and `scan` operations (in a specified proportion), with target keys selected according to one of several distributions (e.g., uniform, zipfian). The `update` queries

are sent only to the master while the `read` and `scan` queries are sent only to the slaves in a round-robin manner. According to previous work [12], and confirmed by our test, these queries are CPU-bound when the database can reside in the memory. For each YCSB query, we assign it an SLA penalty cost function and decide the X_q value the same way as we assign to TPC-W query.

Most of the following results come from TPC-W benchmark. The results of Yahoo Cloud Serving Benchmark are very similar and also support our conclusion. Due to the space limits, they can be found in the supplementary file.

3 System Modeling Module

In this section, we describe the system modeling module in SmartSLA. The system modeling module is designed for solving the problem of "How to accurately predict the average SLA penalty cost under different resource allocations?".

3.1 Overview of System Modeling Module

The core structure of our system modeling module is based on machine learning technologies, as shown in Figure 4. The appeal of machine learning techniques is that they are *data-driven*, i.e., the *resulting models* may be different for different systems, but the *modeling process* is the same.

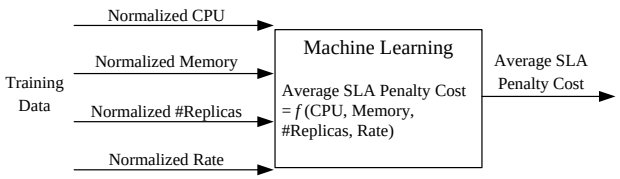


Fig. 4: The overview of system modeling module

We choose CPU share, memory size, client query arrival rate, and replica number as the features for machine learning models. These features not only contain the controllable knobs such as CPU share, memory size, and replica number, but also some uncontrollable but observable factors such as client query arrival rate. Note that, clients' queries are similar and usually follow specific template and mix for a DBaaS provider. For example, Salesforce.com's Force.com (www.salesforce.com) provides DBaaS for their tenants, mainly in customer relationship management (CRM) area. Due to the similarity among CRM applications, the queries are also similar and usually follow specific template and mix. With all this in mind, we assume that the queries follow specific template and mix of TPC-W.

For the machine learning algorithms, we use an off-the-shelf WEKA package [22]. We investigate two classes of machine learning techniques, i.e., parametric learning techniques and non-parametric ones. For parametric learning techniques, we start with a simple *linear regression* model. However, the accuracy turns out to be unsatisfactory, because of the nonlinear relationship between the SLA penalty cost and some of the resources such as memory size. Then, we find that the prediction accuracy can be improved by models such as the *regression tree* model [27], which take the nonlinearity into account. To further improve the prediction accuracy, we use a boosting approach, called *additive regression* method [18], with regression tree as the underlying weak learner. For the non-parametric learning techniques, we investigate the k-nearest neighbor (k-NN) model [16]. Instead of performing explicit generalization, k-NN directly compares new instances with instances seen in training.

TABLE 2: Prediction error of learning algorithms.

Algorithms	RMSE	RAE	Building Time
Linear Regression	0.0632	44.0%	110ms
k-NN ($k = 2$)	0.0718	48.3%	15ms
k-NN ($k = 11$)	0.0555	37.3%	15ms
k-NN ($k = 20$)	0.0577	39.2%	15ms
Regression Tree	0.0555	37.3%	313ms
Boosting	0.0459	28.6%	1156ms

3.2 Comparison of the Learning Algorithms

In Table 2, we report the error, in terms of the root mean squared error (RMSE) and the relative absolute error (RAE)[5]. We study the reason why different learning algorithms give different errors. In our previous study [35], we find that some parameters, such as the memory size, impact the average SLA cost in a strongly nonlinear way. Although increasing memory can help reduce the average SLA penalty cost initially, when memory is larger than certain threshold, e.g., larger than 512MB, adding more memory does not help further. This phenomenon is due to the cache effect—the database of TPC-W is rather small, only about 280MB, and so when the memory size is large enough, memory stops to be a main bottleneck. Thus, the relation between the average cost and memory size is not linear as we expect. Therefore, for a linear model, such as linear regression, because it assumes that the parameters (resource allocation) affect the output (average SLA penalty cost) in a linear fashion, it has large errors. The non-linear models, such as regression tree and boosting approach, have smaller errors.

In addition, we also collect the overhead of the learning algorithms in terms of the time spent. It consists of two parts: (1) the time for building the corresponding models, and (2) the time for using the built models to make a prediction on a new data point. From Table 2, we can see that as we use more advanced machine learning algorithms, the time overhead for building the complicated models increases. However, because model building is a one-time process and can be done offline, such an overhead is not a serious issue. For the algorithm to be applicable online, the more important factor is how quickly the algorithms can make the prediction for a newly obtained data point, i.e., their evaluation time. Based on our experimental results, all the algorithms under study can make a new prediction in less than 1ms, which is an acceptable overhead for our application (as will be shown in later sections).

5. The relative absolute error is defined as $\frac{1}{\Omega}\sum_{s=1}^{\Omega}|p_s - t_s|/t_s$, where Ω is the total number of samples, p_s is the prediction on the s-th sample and t_s is the ground truth of the s-th sample.

The details of training data collection and different learning algorithms are shown in Section 1 of the supplementary file. By comparing all the learning algorithms in the aspects of prediction errors and overheads, we choose to use boosting approach due to its smallest prediction errors and reasonable overheads.

3.3 Adding Confidence Interval to Learning

Although we take the model with the smallest prediction error, there are still two main problems for the machine learning techniques. First, the error for the model is partly due to the complex system state space, and this is because the running time of queries is not necessarily determined only by CPU share and memory share. For example, queries that belong to the same query type (e.g., Search Request) may have different cardinality and also different running neighbors in the run time. While we have captured a significant number of features in the study, the magnitude of the error indicates that more features such as the ones mentioned above may be relevant. Second, all the above models give a single point prediction, i.e., they return the predicted SLA penalty cost. The single point estimation gives no clue of the confidence on the estimation.

In order to obtain a confidence interval of the prediction, we propose a committee based machine learning model which is built as follows.

1) We use the total training data points as the input, which is the same as the previous single point estimation.
2) We build a committee which composes of η members. Each member randomly choose a part, i.e., ξ of the total training data points as the training data where $0 < \xi < 1$.
3) Each member will give a single point prediction, i.e., AC_i where $i = 1, 2, ..., \eta$.
4) We take the average of AC_i as the final prediction and the standard deviation of AC_i as the error of the prediction.

According to statistical learning theories [16], the variance among the predictions of the committee members serves as a good estimation on the variance of the prediction error of our machine learning method. We evaluate our committee-based learning model in Section 6.5.

4 From Single Client to Multiple Clients: Interference Analysis

In the previous sections, we have built prediction models for the average SLA penalty cost under different resource allocations, of a *single* client. However, the final target is to use such models for resource management among *multiple* clients, where there may exist interference among clients sharing the same host. In order for our models built for individual clients to be applicable to the case where multiple clients share the same host, a necessary condition is that the interference among the clients is relatively small. In this section, we investigate the level of interference among clients and the factors that influence such interference.

Although modern virtual machine technology provides strong isolation among virtual domains, e.g., security isolation prevents a malicious application in one virtual machine from attacking applications or accessing data in other virtual machines, it does not provide effective performance isolation. More specifically, while the hypervisor (a.k.a. the virtual machine monitor) can allocate different resources shares to different virtual machines, the behavior of one virtual machine can still affect the performance of another adversely due to the shared use of resources in the system. We rely on the CPU/memory allocation and isolation of the underlying virtualization software – Xen in our case. Therefore, in this study we verify that Xen actually provides a clear service differentiation between clients with respect to the CPU shares. We perform experiments to show the interference among different clients with respect to the system resources (mainly CPU) is small for our workload.

4.1 Interference Analysis Settings

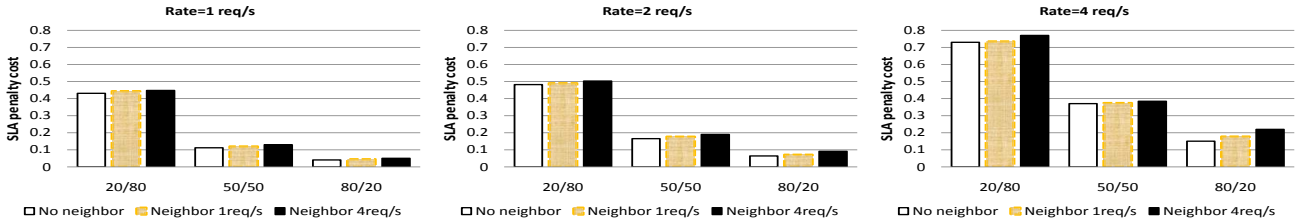
We conduct the following tests for interference analysis. We use the same TPC-W benchmark with Ordering mix as in Section 3. We have two clients, i.e., a foreground one and a background one (a.k.a., neighbor to the foreground one in Figure 5). Because the running time of foreground client varies, we ensure that the background client stays active before the foreground client completes.

We compare the average SLA penalty cost of the foreground client in three cases. In the first case, the foreground client is the only one occupying the resources. In the second case, we set the rate for the background client as 1 req/s and let it share the resources with the foreground client according to the CPU shares. In the third case, we set the rate for the background client as 4 req/s and let it share the resources with the foreground client according to the CPU shares. We also try different CPU share allocations, i.e., 20/80, 50/50 and 80/20 in the experiments. For example, 20/80 denotes that 20 and 80 shares of CPU are allocated to the foreground and background clients, respectively.

4.2 Interference Analysis Result

The experiment results for all the cases with respect to different CPU share allocations are shown in Figures 5(a), 5(b), 5(c), respectively. We have several observations: (1) For the same CPU shares, the average SLA penalty cost will increase when the rate for the foreground client increases from 1 req/s to 4 req/s. (2) For the same rate for the foreground client, the average SLA penalty cost will decrease when the share for the foreground client increases from 20 to 80. These two observations are consistent with the result from Section 3, e.g., the linear regression model. (3) For the same CPU shares and the same rate for the foreground client, the average SLA penalty cost will increase when the background client appears. Moreover, for the same CPU shares and the same rate for the foreground client, the average SLA penalty cost will increase when the rate for the background client increases from 1 req/s to 4 req/s. (4)The experiments show that the average SLA

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2319095, IEEE Transactions on Parallel and Distributed Systems



(a) Arrival rate=1 req/s for foreground client (b) Arrival rate=2 req/s for foreground client (c) Arrival rate=4 req/s for foreground client

Fig. 5: Interference among multiple classes of clients, with different arrival rate for foreground client.

penalty cost difference is small, with less than 0.02 in most of the cases. This is mainly due to the CPU-bound queries that we use. The result shows that the statistical analysis for a client is valid even when the client is sharing the resources with others. (5)There are also extreme cases when the difference is around 0.069, e.g., when the rate for both the foreground and background clients is 4 req/s and the share distribution is 80/20 as shown in Figure 5(c). In summary, the experimental result shows that the statistical analysis for a client is valid even when the client is sharing the resources with others. However, in our future work, we plan to evaluate the effect of interference when using more than two clients and build a model that takes into consideration of performance difference following [21].

5 Resource Allocating Module

In this section, we present the resource allocating module in SmartSLA. The resource allocating module is designed for solving the problem of "How can we allocate resource in order to achieve the minimum cost?".

The core structure of resource allocating module is shown in Figure 6. Firstly, the cost models and the optimization objectives are input to the resource allocating module. The runtime workloads information for clients are also sent from the server to the module. Secondly, the module makes the decision of the next-interval resource allocation. Finally, the module enforces the resource allocation.

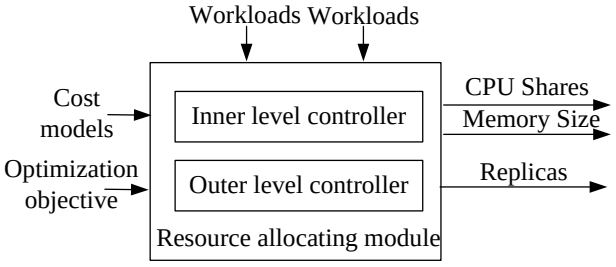


Fig. 6: Overview of resource allocating module

We will use the notations in Table 3 in our discussion. Basically, we use two indices: k means the k-th time interval and i means the i-th class of clients (where i is between 1 and N). In the following two subsections, we first describe the process of grid based searching and multi-level enforcement of the next-interval resource allocation and then present different cost models and optimization objectives that drive the searching and enforcement.

TABLE 3: Notations

k	Interval (180s)
T	The length of the interval
$I(k)$	Infrastructure cost
$M(k)$	The number of replicas
N	Class of clients
$SLA(k)$	Weighted total SLA penalty cost during the k-th interval
q	Query, where q_{start} is the start time of the query q_{time} is the response time of the query
$P(q)$	SLA penalty cost function for query q
$w(i)$	Weight for the i-th class of client
$P_w(q,i)$	weighted SLA penalty cost for the i-th class of clients
$L(i,k)$	The total number of queries for the i-th class of clients during the k-th interval
$cpu(i,k)$	The cpu shares for the i-th class of clients during the k-th interval
$mem(i,k)$	The memory shares for the i-th class of clients during the k-th interval
$AC(i,k)$	The average SLA cost for the i-th class of clients during the k-th interval

5.1 Search and Enforcement

SmartSLA decides the next-interval resource allocation for each client when the previous time interval ends. We use a 3-D array, i.e., $d_i = [d_{i1}, d_{i2}, d_{i3}]$ to denote the difference between the next-interval resource allocation and the current resource allocation for the i-th client. This difference is also denoted as a direction as shown in Figure 7. The first two dimensions, i.e., d_{i1} and d_{i2} denote the directions for CPU and memory tuning respectively while the third dimension d_{i3} is used for replica tuning. We use $D = [d_1, d_2, ..., d_N]$ to denote the global decision for N clients. Since the clients share the machines, we define the third direction as $D_3 = d_{13} = d_{23} = ... = d_{N3}$.

We define $cpu(i,k-1)$, $mem(i,k-1)$, $M(k-1)$ as the cpu shares, memory shares and the number of replicas for the i-th class of clients during the $(k-1)$-th interval. Following the definition of direction, we will enforce the cpu shares, memory shares and the number of replicas for the i-th class of clients during the k-th interval to be:

$$cpu(i,k) = cpu(i,k-1) + d_{i1}$$
$$mem(i,k) = mem(i,k-1) + d_{i2}$$
$$M(k) = M(k-1) + D_3$$

5.2 Cost Models

We show different cost models and optimization objectives that are used to drive the search for the next-interval resource allocation in this subsection.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TPDS.2014.2319095, IEEE Transactions on Parallel and Distributed Systems

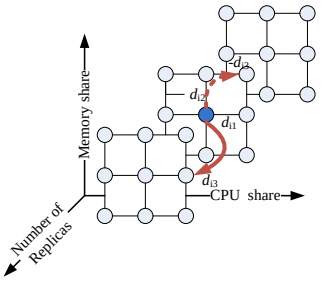


Fig. 7: Searching directions.

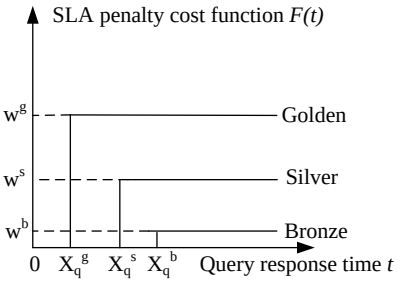


Fig. 8: Weighted SLA penalty cost.

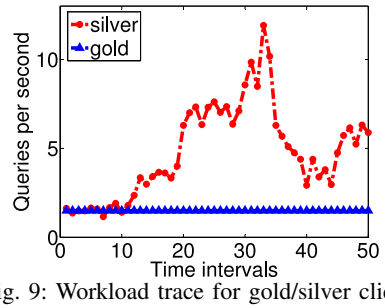


Fig. 9: Workload trace for gold/silver clients.

5.2.1 Total weighted SLA penalty cost model

Assume that there are N classes of clients. For the i-th class of clients we need an SLA penalty cost function $P(q,i)$ which depends on the deadline X_q^i. For the i-th class of clients, we use a weight $w(i)$ to denote the penalty when query q misses the deadline. For example, as shown in Figure 8, assume that we have gold, silver, and bronze clients, then the deadlines for them are X_q^g, X_q^s and X_q^b, respectively and the SLA penalty costs for them are w^g, w^s and w^b, respectively.

We define the weighted SLA penalty cost function as:

$$P_w(q,i) = P(q,i) \times w(i)$$

Within the k-th interval, we define the average SLA penalty cost for the i-th class of clients as the total SLA penalty cost over the total number of queries $L(i,k)$ during the interval.

$$AC(i,k) = \frac{1}{L(i,k)} \sum_{T\times(k-1)\le q_{start}<T\times k} P(q,i)$$

Given that there are N classes of clients, we have the total weighted SLA penalty cost for the k-th interval as

$$\begin{aligned} SLA(k) &= \sum_{i=1}^{N} \sum_{T\times(k-1)\le q_{start}<T\times k} P_w(q,i) \\ &= \sum_{i=1}^{N} AC(i,k) \times L(i,k) \times w(i) \end{aligned} \quad (1)$$

If the optimization objective is to minimize the total weighted SLA penalty cost, then the problem can be formalized as follows.

$$\textit{next direction} = \arg\min_{D} \textit{SLA}(k)$$

Here $SLA(k)$ follows (1) where $AC(i,k) = f_i(cpu(i,k-1)+d_{i1}, mem(i,k-1)+d_{i2}, M(k-1)+D_3, L(i,k))$ is the model that is learned by machine learning techniques. $\sum_{i=1}^{N} d_{i1} = 0$, $\sum_{i=1}^{N} d_{i2} = 0$ and $D_3 = 0$.

5.2.2 Infrastructure cost model

From a service provider's point of view, infrastructure cost may involve many factors: hardware, software, DBA expenses, electricity bills, etc. For example, the price table from Amazon Relational Database Service[6] shows that they adopt a simple linear model between the infrastructure cost and the number of machines. We define c as the cost per node per interval. Thus we have $I(k)$ to be proportional to the number of database replicas $M(k)$ as $I(k) = c \times M(k)$. Then the infrastructure cost for direction D can be calculated as

$$I(k) = I(k-1) + c \times D_3 \quad (2)$$

If the optimization objective is to minimize the total weighted SLA penalty cost plus infrastructure cost, then the problem can be formalized as follows.

$$\textit{next direction} = \arg\min_{D} \textit{SLA}(k) + I(k)$$

Here $SLA(k)$ follows (1), $I(k)$ follows (2), $\sum_{i=1}^{N} d_{i1} = 0$, $\sum_{i=1}^{N} d_{i2} = 0$, and $D_3 \in \{-1, 0, 1\}$ as we want to add/remove at most one replica during one interval. We give this limitation since adding a replica may involve an action cost as will be shown later.

5.2.3 Action cost model

In order to model the action cost involved in adding replicas, we use $\overline{D}$ to denote the semi-reverse direction of D where we have $\overline{d_i} = [d_{i1}, d_{i2}, -d_{i3}]$. Compared with D, the first and the second dimensions are the same but the third dimension is opposite in $\overline{D}$. Similarly, we have

$$\overline{SLA}(k) = \sum_{i=1}^{N} \overline{AC}(i,k) \times L(i,k) \times w(i)$$

Here $\overline{AC}(i,k) = f_i(cpu(i,k-1)+d_{i1}, mem(i,k-1)+d_{i2}, M(k-1)-D_3, L(i,k))$. Then we can model the action cost $A(k)$ during the scale-out process ($D_3 = 1$) and scale-in process ($D_3 = -1$) as

$$A(k) = \begin{cases} \frac{\alpha t_{\text{action}}}{T}(\overline{SLA}(k) - SLA(k)) & : \text{if } D_3 = 1 \\ 0 & : \text{otherwise.} \end{cases} \quad (3)$$

For a scale-out process which lasts for t_{action}, we introduce a parameter α as an amortization factor. α indicates the confidence of the controller in adding a new replica as a "future investment". When the amortization factor is low,

6. (Nov. 8, 2010) Small DB Instance: 1.7 GB memory, 1 ECU (1 virtual core with 1 ECU), 64-bit platform, Moderate I/O Capacity is $0.11 per hour; Large DB Instance: 7.5 GB memory, 4 ECUs (2 virtual cores with 2 ECUs each), 64-bit platform, High I/O Capacity is $0.44 per hour.

e.g., when $\alpha = 0$, the action cost will be distributed along the infinity intervals. In this case, SmartSLA is optimistic about the future intervals and it does not consider the action cost at all. However, when the amortization factor is high, e.g., $\alpha = 1$, SmartSLA is pessimistic about the future intervals and adds a new replica only when the action cost can be compensated in the next interval. For scale-in process, stopping an existing replica only needs to wait for the current query request to this replica to finish and its cost is almost 0.

If the optimization objective is to minimize the total weighted SLA penalty cost plus infrastructure cost plus action cost, then the problem can be formalized as follows.

$$\textit{next direction} = \arg\min_{D} \textit{SLA}(k) + I(k) + A(k)$$

Here $\textit{SLA}(k)$ follows (1), $I(k)$ follows (2), $A(k)$ follows (3), $\sum_{i=1}^{N} d_{i1} = 0$, $\sum_{i=1}^{N} d_{i2} = 0$, and $D_3 \in \{-1, 0, 1\}$.

6 Evaluation

In this section, we present the evaluation results of resource allocation module when different cost models are used.

6.1 Clients, Workloads, SLAs and Weights

In all our experiments, we use two classes of clients, the gold clients and the silver ones. Both of their arrival processes are Poisson process and both of the request mixes are TPC-W ordering. For the silver clients, we use a real workload trace as shown in Figure 9 to evaluate the system under control. The workload trace is generated based on the Web traces from the 1998 World Cup site [4]. We scale down the rates to fit our experiment environment, e.g., we use 1 req/s to mimic 10k rate. For the gold clients, we use a constant workload of 1.56 req/s as shown in Figure 9, which is the average arrival rate of the silver client in the first 10 time intervals (before the burst happens). For simplicity we set the same deadline for the gold and silver clients, i.e., $X_q = X_q^g = X_q^s$. Then we have the same machine learning model for them accordingly. However, we set different weights, i.e., $w^g = 0.2$ and $w^s = 0.1$ for gold and silver clients, respectively. When a request from gold/silver clients misses the deadline, the service provider will pay 0.2/0.1, respectively.

The initial CPU shares and memory shares are set to 50 and 512MB respectively. Each experiment runs over 50 intervals, i.e., 9000 seconds. The warmup and the cool-down times for each experiment are both 180 seconds. Each experiment is repeated 5 times and the average of them is reported as the result.

6.2 Evaluation with Weighted SLA Penalty Cost Model

We conduct the experiments to minimize the total weighted SLA penalty cost with different number of database replicas, i.e., $M(k) = 2, 3, 4, 5$. For comparison purpose, we set CPU shares and memory shares for each client to 50 and 512MB as baseline. Figure 10 shows the total weighted SLA penalty cost for the baseline and SmartSLA. We can get two observations from the figure. (1) As the we add replicas from 2 to 5, in both cases the total weighted SLA penalty cost decreases. This is expected as more replicas will decrease the SLA penalty. (2) Compared with the baseline, SmartSLA always reduces cost. This result verifies that SmartSLA can adaptively allocate resources considering both the request rates and also the weights. More details of evaluation with weighted SLA penalty cost model, e.g., the weighted SLA penalty cost over time for the baseline and SmartSLA, the percentage of total CPU and memory that is allocated dynamically to the gold client are shown in Section 2 of the supplementary file.

TABLE 4: Comparison of cost with/without consideration of infrastructure cost

Number of replicas	Weighted SLA penalty Cost	Infrastructure Cost (Avg. # of replicas)	Total Cost
2	2363	100(2)	2463
3	2154	150(3)	2304
4	2115	200(4)	2315
5	2080	250(5)	2330
SmartSLA	2097	187(3.74)	2284

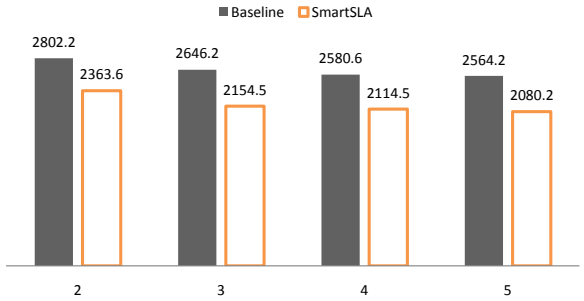


Fig. 10: Total weighted SLA penalty cost with number of replicas=2,3,4,5.

6.3 Evaluation with Infrastructure Cost Model

We conduct the experiments to minimize the sum of weighted SLA penalty cost and infrastructure cost. Table 4 shows the weighted SLA penalty cost, infrastructure cost, and total cost when $c = 1$. The first 4 rows are the SmartSLA results with *fixed* number of replicas (2 to 5). In the last row we show the SmartSLA results when SmartSLA is allowed to automatically decide and tune the best number of replicas to use in each time interval. From the results we can obtain the following observations. (1) When more replicas are used, the SLA penalty cost decreases as expected. But if we take the infrastructure cost into consideration as well, then a larger number of replicas is not necessarily more cost effective. (2) When the replica number is allowed to change dynamically, SmartSLA fuses the infrastructure cost with the system model and makes intelligent decisions on the number of replicas to use in each time interval and it achieves lower cost than *any* of the cases with fixed number of replicas.

6.4 Evaluation with Action Cost Model

We conduct the experiments to minimize the sum of weighted SLA penalty cost, infrastructure cost and action

cost. Table 5 summarizes the costs with amortization factor α =0, 0.1, and 1. Note that, the objective of this evaluation is to show that the total cost could be further reduced if action cost is taken into consideration rather than to find out the best value of amortization factor. As can be seen, both too aggressive ($\alpha = 0$) and too conservative ($\alpha = 1$) settings give relatively higher cost, while with $\alpha = 0.1$, the total cost is further reduced.

TABLE 5: Comparison of cost with different amortization factors.

Amortization factor	Weighted SLA penalty Cost	Infrastructure Cost (Avg. # of replicas)	Total Cost
0	2151	408(2.72)	2559
0.1	2130	378(2.52)	2508
1	2364	300(2)	2664

6.5 Evaluation with Confidence Interval

We evaluate the committee-based learning model with confidence interval by comparing the real weighted SLA penalty cost with the predicted one, which is the average weighted SLA penalty cost with standard deviation. We show the real and the estimated weighted SLA penalty cost for total(gold+silver), silver and gold clients in Figure 11 when we use the weighted SLA penalty cost model and the number of replicas is fixed at 2. We can see that the weighted SLA penalty cost for total, silver and gold clients are all within the boundary of "Mean−Std" and "Mean+Std". From this result we can see that such information on error bounds will help the DBaaS provider to make decisions with confidence.

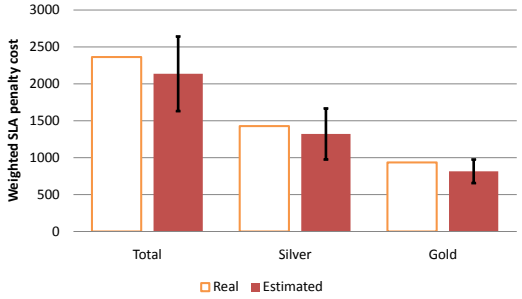


Fig. 11: Real vs. estimated weighted SLA penalty cost.

6.6 Evaluation with YCSB benchmark

We present the system modeling module and the resource allocating module evaluation for YCSB benchmark in Sections 3 and 4, and compare it with TPC-W benchmark in Section 5 in the supplementary file. The similarity is that, (1) non-linear learning algorithms have smaller prediction errors than linear regression; and (2) our proposed SmartSLA is also able to help DBaaS providers to reduce cost when they are servicing tenants running YCSB benchmarks. There is also difference between the two benchmarks. For example, the relative absolute error of linear regression for TPC-W is 44% while it is much higher for YCSB (59.5%). Moreover, there is more fluctuation of tuning of CPU in YCSB than in TPC-W when we compare Figure 3 with Figure 7 in the supplementary file. This is mainly due to the significant different characteristics of the two benchmarks. TPC-W is one of a traditional relational database benchmarks while the original YCSB benchmark is built for evaluating the performance of different "key-value" and "cloud" serving stores.

7 Related Work

From a general system point of view, most of the related work can be broadly classified into dynamic resource provisioning and allocation of CPU [33], cache and memory [9], [23], [25], storage [3], [20], [24], network [33] and energy [7]. For example, Urgaonkar *et al.* [33] present techniques for provisioning CPU and network resources in shared hosting platforms. Lu *et al.* dynamically adjust the cache size [25] while Chou *et al.* manage the buffer pool of a relational database management system [9]. Chase *et al.* manages energy and server resources in hosting centers [7]. Albrecht *et al.* propose Janus to provision flash storage in the Cloud [3]. Rai *et al.* present a generic and extensible tool that can be used to solve resource allocation problems [28].

Most of the early work assumes that the system under control is linear, and that the parameters can be identified offline [14]. However, there will be oscillation and the system will be unstable once the operation point moves out of the linear area. Compared with the previous work, we use machine learning technique to build the relationship based on tree models. One of the benefits of using a tree model is to overcome the non-linear obstacle. Note that Ganapathi *et al.* [19] and Mozafari *et al.* [26] also use a machine learning technique called KCCA to predict metrics such as elapsed time, records used, disk I/Os, etc. Compared with KCCA, which is sensitive to some modeling parameters such as the definition of the Euclidean distance and the scale factor, we focus on popular and easy-to-use techniques such as linear regression and boosting.

From a DBMS point of view, SQL queries can be very complex and resource intensive. The same query when executed with different parameters can consume vastly different amount of resources and take much more/less time than another instance of the same query (even when run in isolation). For example, Duggan *et al.* [17] and Ahmad *et al.* [2] explore the interaction among concurrent execution queries. Compared with their work which focuses on OLAP workloads, we are focusing on the OLTP workloads. We are planning to take into consideration of the interaction among concurrent execution queries for our prediction in the future. The work presented in Soror *et al.* [31] is most closely related to ours. There are two significant differences. (1) They model the problem as a service differentiation problem under the resource constraints. However, we model the problem as a two level optimization/control problem. (2) They model the relationship between the performance and the system metrics such as CPU and memory individually. However, in our work, we combine the system metrics, the number of replicas, and the arrival rate as the multiple

input. Consequently our model is comprehensive to capture various relationships among system metrics and average SLA penalty cost. Moreover, we also consider the action cost related to database systems and provide a model for replica tuning.

8 Conclusion

In this paper, we propose SmartSLA, a cost-sensitive virtualized resource management system for CPU-bound database services. SmartSLA minimizes the total cost for a DBaaS provider which is composed of SLA penalty cost, infrastructure cost and action cost. SmartSLA consists of two main components: a system modeling module and a resource allocating module. SmartSLA is evaluated by using the TPC-W and YCSB benchmarks with dynamic workload trace and multiple database tenants. The experimental results show that SmartSLA is able to minimize the total cost under time-varying workloads compared to the other cost-insensitive approaches.

There are several limitations in our framework. (1) Although we evaluate SmartSLA with both TPC-W and YCSB benchmarks, a DBaaS provider may encounter changes which correspond to new data points that are not part of the training phase. These changes are discussed in more details in Section 6 in the supplementary file. We plan to implement a feedback channel where the real-time information of SLA penalty cost and the resource allocation is sent back to the system modeling module. This feedback information can further help the prediction module to improve the accuracy of its future predictions by introducing new training data [15]. (2) In a virtualized environment, CPU and memory sharing adds-up linearly, but IO does not. The current work does not consider the sharing of IO because the database will reside in the main memory as a major trend for OLTP applications [32]. We would like to include IO as another feature in our machine learning model to deal with OLAP applications as our future work.

References

[1] Transaction processing performance council. tpc benchmark w (web commerce). number revision 1.8, february 2002.

[2] M. Ahmad, A. Aboulnaga, S. Babu, and K. Munagala. Interaction-aware scheduling of report-generation workloads. *The VLDB Journal*, 20:589–615, Aug. 2011.

[3] C. Albrecht, A. Merchant, M. Stokely, M. Waliji, F. Labelle, N. Coehlo, X. Shi, and E. Schrock. Janus: Optimal flash provisioning for cloud storage workloads. In *Proc. of USENIX ATC*, 2013.

[4] M. Arlitt and T. Jin. Workload characterization of the 1998 world cup. web site. *HP Tech. Rep.*, 1999.

[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. of SIGMOD*, 2008.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *Proc. of SOSP*, 2003.

[7] J. S. Chase, D. C. Anderson, P. N. Thakar, A. M. Vahdat, and R. P. Doyle. Managing energy and server resources in hosting centers. In *Proc. of SOSP*, 2001.

[8] Y. Chi, H. J. Moon, and H. Hacigümüş. icbs: incremental cost-based scheduling under piecewise linear slas. In *Proc. of VLDB*, 2011.

[9] H.-T. Chou and D. J. DeWitt. An evaluation of buffer management strategies for relational database systems. In *Proc. of VLDB*, 1985.

[10] B. F. Cooper, A. Silberstein, E. Tam, R. Ramakrishnan, and R. Sears. Benchmarking cloud serving systems with ycsb. In *Proc. of SoCC*, 2010.

[11] C. Curino, E. P. Jones, S. Madden, and H. Balakrishnan. Workload-aware database monitoring and consolidation. In *Proc. of SIGMOD*, 2011.

[12] S. Das, S. Nishimura, D. Agrawal, and A. E. Abbadi. Albatross: Lightweight elasticity in shared storage databases for the cloud using live data migration. In *Proc. of VLDB*, 2011.

[13] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *Proc. of SOSP*, 2007.

[14] Y. Diao, N. Gandhi, J. Hellerstein, S. Parekh, and D. Tilbury. Mimo control of an apache web server: Modeling and controller design. In *Proc. of ACC*, 2002.

[15] S. Duan, V. Thummala, and S. Babu. Tuning database configuration parameters with ituned. In *Proc. of VLDB*, 2009.

[16] R. O. Duda, P. E. Hart, and D. G. Stork. *Pattern Classification*. Wiley, New York, 2nd edition, 2001.

[17] J. Duggan, U. Cetintemel, O. Papaemmanouil, and E. Upfal. Performance prediction for concurrent database workloads. In *Proc. of SIGMOD*, 2011.

[18] J. Friedman. Stochastic gradient boosting. 1999.

[19] A. Ganapathi, H. Kuno, U. Dayal, J. L. Wiener, O. Fox, and M. Jordan. Predicting multiple metrics for queries: Better decisions enabled by machine learning. In *Proc. of ICDE*, 2009.

[20] A. Gulati, G. Shanmuganathan, X. Zhang, and P. Varman. Demand based hierarchical qos using storage resource pools. In *Proc. of USENIX ATC*, 2012.

[21] D. Gupta, L. Cherkasova, R. Gardner, and A. Vahdat. Enforcing performance isolation across virtual machines in xen. In *Proc. of Middleware*, 2006.

[22] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. In *Proc. of the 5th Australian Joint Conference on Artificial Intelligence*, volume 11, 2009.

[23] J. Hwang and T. Wood. Adaptive performance-aware distributed memory caching. In *Proc. of USENIX ICAC*, 2013.

[24] D. Li, X. Liao, H. Jin, B. B. Zhou, and Q. Zhang. A new disk i/o model of virtualized cloud environment. *IEEE Trans. Parallel Distrib. Syst.*, 24(6), 2013.

[25] Y. Lu, T. Abdelzaher, and A. Saxena. Design, implementation, and evaluation of differentiated caching services. *IEEE Transactions on Parallel and Distributed Systems*, 15(5), 2004.

[26] B. Mozafari, C. Curino, A. Jindal, and S. Madden. Performance and resource modeling in highly-concurrent oltp workloads. In *Proc. of SIGMOD*, 2013.

[27] R. J. Quinlan. Learning with continuous classes. In *Proc. of the 5th Australian Joint Conference on Artificial Intelligence*, 1992.

[28] A. Rai, R. Bhagwan, and S. Guha. Generalized resource allocation for the cloud. In *Proc. of SOCC*, 2012.

[29] B. Reinwald. Database support for multi-tenant applications. In *Proc. of WISS*, 2010.

[30] B. Schroeder, M. Harchol-Balter, A. Iyengar, E. M. Nahum, and A. Wierman. How to determine a good multi-programming level for external scheduling. In *Proc. of ICDE*, 2006.

[31] A. A. Soror, U. F. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath. Automatic virtual machine configuration for database workloads. In *Proc. of SIGMOD*, 2008.

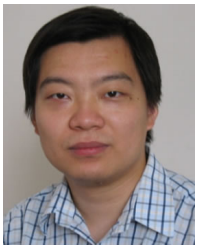
[32] M. Stonebraker, S. Madden, D. J. Abadi, S. Harizopoulos, N. Hachem, and P. Helland. The end of an architectural era: (it's time for a complete rewrite). In *Proc. of VLDB*, 2007.

[33] B. Urgaonkar, P. Shenoy, and T. Roscoe. Resource overbooking and application profiling in shared hosting platforms. In *Proc. of OSDI*, 2002.

[34] C. D. Weissman and S. Bobrowski. The design of the force.com multitenant internet application development platform. In *Proc. of SIGMOD*, 2009.

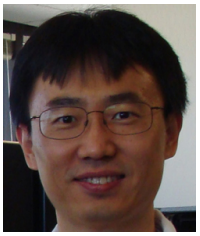
[35] P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, and H. Hacigumus. Intelligent management of virtualized resources for database systems in cloud environment. In *Proc. of ICDE*, 2011.

[36] F. Yang, J. Shanmugasundaram, and R. Yerneni. A scalable data platform for a large number of small applications. In *Proc. of CIDR*, 2009.



Pengcheng Xiong received the B.S. degree in automatic control from Huazhong University of Science and Engineering, Wuhan, China, in 2003. He received the M.S. and Ph.D. degree in control theory and engineering from the Department of Automation, Tsinghua University, Beijing, China, in 2008. He received the Ph.D. degree in computer science from the School of Computing, Georgia Institute of Technology, Atlanta, USA, in 2012.

He is currently a research staff member at data management department of NEC Labs America, Cupertino, USA. His research interest includes Cloud computing, Data management systems, Data center resource management and Service oriented computing. He has already published a dozen of papers in journals and conference proceedings.



Yun Chi is currently a Research Staff Member at NEC Laboratories America. He received his Ph.D. degree in Computer Science from University of California, Los Angeles, in 2005. His primary research interests include databases, cloud computing, social networks, data mining, machine learning, and information retrieval.

Shenghuo Zhu is currently a Senior Research Staff Member in NEC Laboratories America. He received his Ph.D degree in Computer Science from University of Rochester in 2003. His primary research interests include machine learning, computer vision, data mining and information retrieval.

Hyun Jin Moon received the Ph.D. degree and M.S. degree in Computer Science from UCLA in 2008 and 2004, and the B.S. in Electrical Engineering and Computer Science from UC Berkeley in 2002. He is currently a software engineer at Google Inc. Before that, he was a research staff member at data management department of NEC Labs. His research interest is in DBMS, data management in the cloud, schema evolution, temporal database, and data integration.

Calton Pu received his PhD from University of Washington in 1986 and served on the faculty of Columbia University and Oregon Graduate Institute. Currently, he is holding the position of Professor and John P. Imlay, Jr. Chair in Software at the College of Computing, Georgia Institute of Technology. He is leading the Infosphere project, building software tools to support information flow-driven applications such as digital libraries and electronic commerce. Infosphere builds on his previous and ongoing research interests. First, he has been working on next-generation operating system kernels to achieve high performance, adaptiveness, security, and modularity, using program specialization, software feedback, and domain-specific languages. This area has included projects such as Synthetix, Immunix, Microlanguages, and Microfeedback, applied to distributed multimedia and system survivability. Second, he has been working on new data and transaction management by extending database technology. This area has included projects such as Epsilon Serializability, Reflective Transaction Framework, and Continual Queries over the Internet. His collaborations include applications of these techniques in scientific research on macromolecular structure data, weather data, and environmental data, as well as in industrial settings. He has published more than 60 journal papers and book chapters, 150 conference and refereed workshop papers, and served on more than 90 program committees, including the co-PC chairs of SRDS'95, ICDE'99, CoopIS'02, SRDS'03, DOA'07, and co-general chair of ICDE'97, CIKM'01, ICDE'06, DEPSA'07, CEAS'07.

Hakan Hacıgümüş received his PhD from University of California, Irvine in 2000. He is currently the head of data management department of NEC Labs America. Before that, he was a research scientist in IBM Research from 2003 to 2009 and a Lead Software Engineer/Researcher in IBM Silicon Valley Lab from 2000 to 2003. His research interest is in Data Management, Cloud Computing, Big Data, Mobility, SOA (Service Oriented Architecture), SaaS (Software-as-a-Service), security and privacy, business intelligence and analytics, information lifecycle management, appliance models, information management middleware, risk management, information and application integration.