Ensemble: A Tool for Performance Modeling of Applications in Cloud Data Centers

Jin Chen, *Member, IEEE*, Gokul Soundararajan, *Member, IEEE*, Saeed Ghanbari, *Member, IEEE*, Francesco Iorio, Ali B. Hashemi, *Member, IEEE*, and Cristiana Amza, *Member, IEEE*

Abstract—We introduce Ensemble, a runtime framework and associated tools for building application performance models on-the-fly. These dynamic performance models can be used to support complex, highly dimensional resource allocation, and/or what-if performance inquiry in modern heterogeneous environments, such as data centers and Clouds. Ensemble combines simple, partially specified, and lower-dimensionality models to provide good initial approximations for higher dimensionality application performance models. We evaluated Ensemble on industry-standard and scientific applications. The results show that Ensemble provides accurate, fast, and flexible performance models even in the presence of significant environment variability.

Index Terms—Performance modeling

1 Introduction

Cloud applications and services are hosted at large data centers, e.g., Google App Engine data center [1] and Amazon Relational Database Service data center [2]. These cloud data centers are highly dynamic environments, co-hosting several applications sharing heterogeneous resources in a constantly evolving environment. In such environments, uncontrolled resource sharing between co-hosted applications often results in performance degradation and violation of service level agreements to applications (SLAs).

State of the art techniques for resource provisioning and capacity planning usually rely on off-line extensive measurements, and/or sysadmin or analyst expertise. To prevent SLAs violations, this may result in over-provisioning for all possible combinations of peak incidental loads, which is unacceptable due to its excessive cost. Moreover, while profiling and monitoring tools [3], [4] help sysadmins to learn about the performance and the resource usage of the system for a runtime configuration parameter setting, sysadmins need to decompose end-to-end SLAs into per-component SLAs. Furthermore, sysadmins need to understand the behavior of the system under various configuration parameter settings and workloads as they present to the system, dynamically, over time.

Per-application *performance modeling* has previously been proposed for *on the fly* adjustment of resource allocation in order to meet per-application SLAs in data centers [5], [6], [7], [8], [9]. Such performance models were previously demonstrated to allow for resource sharing between applications without incurring SLA violations for traditional applications running in a fully controlled data center. Performance models have also been shown useful for automating resource allocation and capacity planning in a heterogeneous cloud, in presence of different resources and workload mixes [10].

In this paper, we address the important problems of optimizing modelling time as well as performance model evolution for complex and highly dynamic systems, such as database and storage systems [2], [11], [12], [13], [14] running in a fully heterogeneous cloud [10].

In such dynamic and complex environments it is not possible to exhaustively evaluate all combinations of configuration parameter settings and workloads at the time of system deployment. Hence, the performance models of systems and applications, as well as the human expertise incorporated into these models are almost always partial, and incrementally evolving over time.

With this paper, we develop a language and run-time integrated framework that allows i) expressing sysadmin hypotheses and guidance for model building and model evolution on the fly and ii) incrementally evolving existing models in new configurations and for new workloads. We show that this integrated framework can enable model evolution by two examples: a database application model, as introduced in our earlier work [15], [16], and a neuroscience application running in a heterogeneous cloud environment.

Towards this, we propose Ensemble, a method to leverage automated black-box modeling, coupled with administrator expertise, wherever available, to build and evolve an ensemble/mixture of models for a variety of applications in heterogeneous clouds. While our approach has some similarities with related work in performance prediction and performance modeling domains, it is unique in three aspects. First, it automatically leverages different types of

- *J. Chen, A.B. Hashemi, and C. Amza are with the Edward S. Rogers Sr. Department of Electrical and Computer Engineering, University of Toronto, Toronto, ON, Canada. E-mail: {jinchen, hashemi, amza}@ece.utoronto.ca.*
- *G. Soundararajan and S. Ghanbari are with NetApp, Inc., Paloalto, CA. E-mail: {gokuls, saeed.ghanbari}@netapp.com.*
- *F. Iorio is with Autodesk Research and Department of Computer Science, University of Toronto. E-mail: francesco.iorio@autodesk.com.*

Manuscript received 1 Aug. 2014; revised 30 Apr. 2015; accepted 11 May 2015. Date of publication 17 Aug. 2015; date of current version 2 Mar. 2016. Recommended for acceptance by K. Keahey, I. Raicu, K. Chard, B. Nicolae. 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 no. 10.1109/TCC.2015.2469656

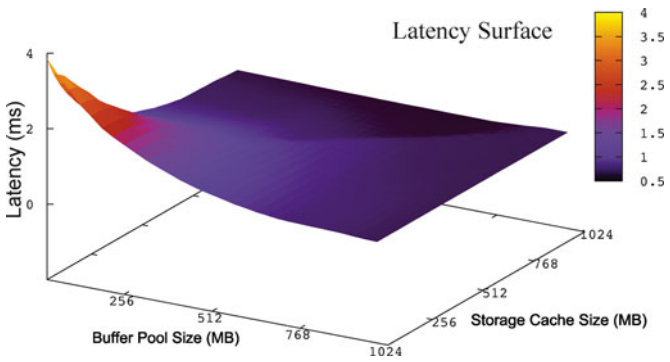


Fig. 1. RUBiS latency surface.

model templates, each of which may be suitable for a subset of all possible operating modes of a system. Second, it reduces the cost of building and evolving performance models by leveraging an intelligent sampling technique. Third, it incorporates sysadmins partial knowledge of the system to build an inexpensive accurate performance model of the system.

We validated Ensemble in two different case studies. In our first case study, we modeled the end-to-end memory access latency of a cloud hosted database. In our second case study, we modeled the performance of a distributed implementation of NPAIRS, a neuroimaging software package for analyzing fMRI data [17], [18] after a major change in the configuration of the cloud infrastructure.

In the rest of the paper, first in Section 2, we provide a detailed background on automatic performance modeling. In Section 3, we explain Ensemble followed by evaluation results in Section 4. Related work is discussed in Section 5. Section 6 contains the conclusion and some proposed future work.

2 Background

A *performance model* is a mathematical function that calculates an estimate of the application performance for a range of configurations. For example, Fig. 1 shows a performance model as a 3D surface for the online auctions application RUBiS [19]. The model provides an estimate of the average memory access latency (i.e. average page access latency measured at the database buffer pool) of a MySQL database engine running RUBiS, for all possible memory quotas in a two-level memory hierarchy, consisting of a buffer pool and a storage.

Automatic performance model building iterates through two steps: (i) gathering experimental samples, and (ii) modeling computation. Gathering experimental samples means actuating the system into a given resource configuration, running a specific application workload on the system and measuring the application performance. Modeling computation involves mathematical interpolation for building the model using available sampling data. While the computational cost of modeling is typically on the order of fractions of seconds, experimental sampling may take months for mapping exhaustively the configuration space of an application with sufficient statistical accuracy. This is due to dynamic effects, for instance, cache warm-up time, which make reliable actuation and sampling extremely expensive even for a single configuration point. For example, for N modeled resources and/or configuration parameters, and M increments of sampling for each resource, an *application performance surface* model would be an $N+1$-dimensional hyperplane with $O(M^N)$ sample points. For our RUBiS example in Fig. 1, due to the cache warm up effect, experimental sampling takes around 15 minutes of measurements at *each* of the 1,024 (32^2) points of the surface. The total sampling takes approximately 11 days. In an enterprise environment, where 64 GB storage caches are common, if we set sampling increments in 1 GB units, total sampling would take two months. It follows that extensive experimental sampling and building fully automated, black-box performance models based on experimental sampling on a live system is too time consuming.

At the other end of the spectrum is the use of analytical models that rely on sysadmin or analyst's semantic knowledge of the system and application [7], [9], [20]. However, these analytical models are precise only for restricted parts of the system, specific application workload mix, or resource configurations. They are sensitive to dynamic changes and require too much domain expertise.

We also observed that calculating a number of potential models from an existing library of models, or fitting known models provided by a system analyst over the collected sampling data is relatively fast compared to the actuating and sampling process itself. Moreover, disk operational laws and cache operational laws (e.g., cache LRU replacement) and disk access patterns [21] (e.g., sequential) are sometimes known beforehand, and can be taken into account to guide performance modeling.

3 Ensemble Framework

Ensemble, our performance modeling framework, provides lightweight sampling and modeling on the fly by connecting user provided model templates to a progressively accumulating library of models over time. We further make the following assumptions about the application, environment, and system API that guide our modeling process.

3.1 Assumptions

We assume that a QoS requirement or SLA is provided, per application, as the *average memory access latency*, and small variations around this average are acceptable.

3.1.1 Stable Patterns for Application and Environment

The hardware and software environment and workload mix for each application deployment are assumed to exhibit *stable periods* when they do not change significantly. We further assume that each application's deployment exhibits recognizable, repeatable patterns of stable periods over time. Input load for any given application may fluctuate during stable periods. However, given the same query mix, and a sufficient resource allocation, the average memory access latency per application is assumed to be relatively stable during a *stable period*. We observed many web applications exhibit non-sequential, but repeatable disk access patterns [21]. Verma et al. also observed that some prevalent workload mixes of Web applications show a daily, weekly or seasonal repetition pattern [22].

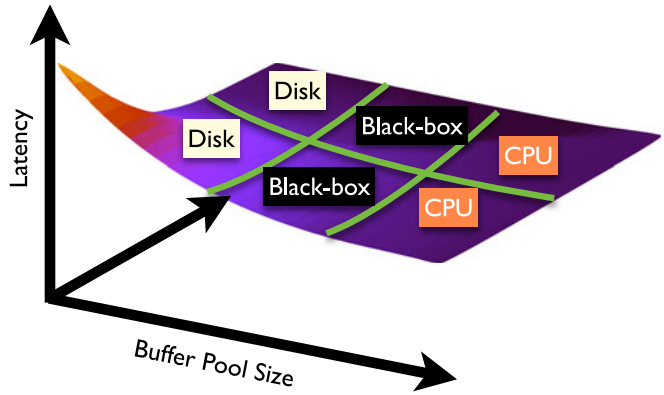


Fig. 2. An example of ensemble learning.

3.1.2 Mechanisms for Enforcing Resource Isolation

Each application is allocated a guaranteed resource quota, per resource, for the entire duration of its execution within a stable period. The application is assumed to execute within a software environment that provides applications with container-based or quanta-based mechanisms for resource isolation at all resources modelled i.e., the CPU, buffer pool, storage cache and storage bandwidth. Specifically, resource controllers are in place to ensure that concurrently running applications cannot interfere with a given application's full use of their resource quotas. For this purpose, we are reusing mechanisms we previously implemented for dynamic partitioning of memory pools in the memory hierarchy and quanta-based I/O scheduling of the storage bandwidth [9].

3.1.3 Analyst Guidance

We assume that the sysadmin or analyst may know relatively simple models with good enough accuracy for parts of the configuration space, or for modeling the operation of certain resources in isolation. For example, the analyst may know that when the application is disk-bound, the relation "the average memory access latency varies on an inverse exponential with the disk bandwidth quanta" applies. This boils down to using a lower dimensionality model in one region of the resource space, instead of the 4D model. A similar function may apply for the cases where the application is CPU-bound.

Fig. 2 depicts a simple scenario for modeling performance of a storage system with Ensemble. We assume that the sysadmin does not know how to model the whole configuration space. However, he or she may suspect that a simple disk model will work in some part of the configuration space that is I/O intensive. We refer the area where a model is valid as the operating region of this model.

Our framework is flexible enough to allow the sysadmin to express model templates, e.g., the belief that a simple disk-bound and/or CPU-bound model may fit parts of a configuration space, and also the operating region(s) of these models, if known. Alternatively, the sysadmin can ask the Ensemble runtime system to automatically find the operating region of any model she provides for validation. Experimental sampling is still required for fitting given model templates to the data and for building black box models from scratch for the regions of the resource or application configuration space where no model is suspected or known. However, specifying suspected models, or reusing older models that may fit parts of the configuration space is expected to speed up total modeling time, as well as to allow interactive model query and validation.

3.2 Model Templates

Ensemble uses *model templates*, wherever available, from a user/analyst, or a repository of models derived previously for other applications. Model templates express analyst beliefs about analytical performance models for the Ensemble to fit or validate with experimental data. The syntax of a model *template* is shown in Listing 1 (keywords are underlined).

```
1 TEMPLATE <templateName>
2 RELATION <relationName> {<metricSet>}
3 CONTEXT {<contextSet>}
```

Listing 1. Syntax of Model Template

Each *template* is identified by a unique name; this allows the template to be saved in a database and later retrieved for future inquiry. The *relation* defines a mathematical function describing the relationship between metrics; it is identified by a relation name and it may be used in several models. A relation could be a suspected mathematical correlation to be validated, curve fitted, or otherwise refined. The *context* is a list of conditions on a set of configurations, in which the analyst believes her template holds. Any parameter, resource configuration, or property that the given relation in the *template* is sensitive to can be specified as an associated (expected) *context* for that relation. Within the template context, configuration ranges for particular resources can be specified, or left as empty. Within any *template*, Ensemble can refer standard math relations, such as, linear, exponential, inverse, machine learning algorithms, as well as non-standard models, that are provided by analysts.

A model template thus expresses incomplete domain knowledge with or without specifying a concrete context. It is the task of the Ensemble runtime to validate and calibrate the associated model, and to find out the context where that model holds.

We list the analytical model templates we will use in this paper in the following. The templates are inspired by our previous modeling experiences. For example, if we look at the RUBiS performance surface (Fig. 1), we notice some characteristics that remind us of memory hierarchy models in general. First, the surface is smooth (a model using "averaging per region" may help to quickly fill in unsampled space based on existing samples). Second, the surface exhibits a plateau loosely corresponding to regions of the memory configuration space where a certain part of the workload fits in memory (a "constant" model would accurately describe such plateau areas and reduce the sampling needed). Finally, wherever there is a bottleneck on a single resource, e.g., disk-bound at low memory configurations, we expect an inverse exponential relationship of

latency with allocations of that one resource, while the model would be largely insensitive to variation of allocation quotas of other resources. In the "middle", the model may be more sensitive to structural artifacts, such as, cache replacement policy and dependency between different components contributing to overall latency; any partial models for each component and rules of composing their latency may help. The above observations are generally applicable to any memory hierarchy model for any application. Based on these intuitions, we define the following approximate models that guide the sampling in Ensemble. The approximate models are the model templates that approximately represent a region of the configuration parameter space of the system.

3.2.1 Basic CPU-Bound Latency Model

The CPU-bound latency model is designed to approximate the region of the configuration space where a workload is CPU-bound as follows:

$$\mathcal{L}_{mem}(\rho_p) = \frac{1}{\rho_p}\mathcal{L}_{mem}(\rho_{p=1}), \tag{1}$$

where ρ_p is the CPU quota allocated to the application and $\mathcal{L}_{mem}(\rho_{p=1})$ is the baseline memory access latency for an application when all the CPU quotas are allocated to the application.

Corresponding to this inverse exponential mathematical relation, a performance model, called CPU-BOUND is presented to the system by the analyst with the syntax shown in Listing 2. The Ensemble run-time learns the model i.e., gathers experimental samples, validates, curve-fits, finds the configuration settings where the relation applies, if any, and computes confidence scores and error rates.

```
1 TEMPLATE CPU-BOUND
2 RELATION Inverse_Exponential(x,y) {
3         x.name='cpu_quota' and
4         y.name='memory_access_latency'
5 }
6 CONTEXT (a) {
7        a.name='memory_size' and a.value='*'
8 }
```

Listing 2. CPU-Bound Model Template

The relation is the Inverse_Exponential relation. This model is declared sensitive to only one parameter: the memory size allocated to the application; the configuration range for which this model may apply is left unknown. Hence, the context in this model is left empty as "*". A more precise context may specify the minimum total amount of buffer pool and storage cache for which the application becomes CPU-bound.

3.2.2 Basic Disk-Bound Latency Model

We assume that the analyst knows that the storage server uses a quanta based scheduler to enforce allocation to the disk bandwidth among multiple applications. Under this assumption, a larger fraction of the disk bandwidth allocated usually leads to lower latencies. Hence, the analyst provides a model template based on inverse exponential, as shown in Listing 3. The inverse exponential is a mathematical relation similar to the one used for defining the CPU-BOUND model, as follows:

$$L_d = \frac{1}{\rho_d}L_{d(\rho_d=1)}, \tag{2}$$

where $L_{d(\rho_d=1)}$ is the *baseline disk latency* for an application, when the whole bandwidth is allocated to that application. This formula is intuitive. For example, if the entire disk was given to the application, i.e., $\rho_d = 1$, then the latency is equal to the underlying disk access latency. On the other hand, if the application is given a small quanta, i.e, $\rho_d \approx 0$, then the storage access latency is very high (approaches ∞).

```
1 TEMPLATE DISK-BOUND
2 RELATION Inverse_Exponential(x,y) {
3         x.name='disk_quota' and
4         y.name='memory_access_latency'
5 }
6 CONTEXT (a) {
7        a.name='memory_size' and a.value='*'
8        b.name='disk_quota' and b.value > 0.1
9 }
```

Listing 3. Disk-Bound Model Template

This model is declared sensitive only to two parameters: the total effective memory size allocated to the application, and the disk quota allocated to the application. The configurations under which the model is considered effective exclude those with very low disk quanta. This is because the analyst knows that, in our storage server, an application's latency shows unacceptably large variation whenever the disk quanta given to the application is less than 32 ms (about 0.1 fraction of the total disk quanta). This is due to insufficient disk access time for the I/O burst, which makes the context switch penalty between applications i.e., the disk seek penalty when switching to servicing the data of a different application, significantly disruptive. The memory size range for which this model may apply is left unknown, as "*".

3.2.3 Memory-Bound Latency Model (A-STOR)

A-STOR is an analytical model for the memory access latency designed for storage intensive workloads. Just like DISK-BOUND, A-STOR ignores the CPU time. However, A-STOR is more sophisticated than DISK-BOUND, because it models the access time to the memory hierarchy in more detail.

Shown in the Listing 4, this model is declared sensitive to three parameters: the MySQL buffer pool size allocated to the application, the storage cache size allocated to the application, and the disk quota allocated to the application.

```
1 TEMPLATE A-STOR
2 RELATION Mem_Latency_Pred(c,s,d,l){
3         c.name = 'mysql_buffer_pool_size' and
4         s.name = 'storage_cache_size' and
5         d.name = 'storage_disk_quanta' and
6         l.name = 'memory_latency'
7 }
8 CONTEXT(){
9 }
```

Listing 4. Memory-Bound Latency Model A-STOR

The relation Mem_Latency_Pred predicts the average memory access latency l per buffer pool page access. It implements the function $\mathcal{L}_{mem}$ derived in Equation (3).

This formula allows the analyst to express the miss rate relationship between the two caches as a function of the cache replacement policy. It is validated by the Ensemble run-time, just like any standard relation. Moreover, this model allows separate modeling of the disk component, which can be plugged into this model as a component of the overall latency. If new sampling needs to occur for the disk component, e.g., the hard disk which we previously sampled for has been replaced with an SSD, then the overall latency model can be recalculated fast, based on resampling only for the changed component, i.e. the disk, and not for the other resources, such as memory and CPU:

$$\mathcal{L}_{mem}(\rho_c, \rho_s, \rho_d) = \underbrace{\mathcal{M}_c(\rho_c)\mathcal{H}_s(\rho_c, \rho_s)L_{net}}_{\text{I/Os satisfied by the storage cache}} + \underbrace{\mathcal{M}_c(\rho_c)\mathcal{M}_s(\rho_c, \rho_s)L_d(\rho_d)}_{\text{I/Os satisfied by the disk}}, \quad (3)$$

where ρ_d is the allocated fraction of disk bandwidth (i.e. disk bandwidth quanta); and ρ_c, ρ_s are the buffer pool, and storage cache quota allocated to the application. The miss/hit ratio at the storage cache, i.e., $\mathcal{M}_s(\rho_c, \rho_s)$ and $\mathcal{H}_s(\rho_c, \rho_s)$, is a function of both the quota at the first level cache (ρ_c), and the quota at the second level cache (ρ_s), while the miss-ratio of the buffer pool, $\mathcal{M}_c(\rho_c)$, is only a function of ρ_c.

Our key idea is to approximate the latency model of a cache hierarchy with the simpler model of a single level of cache, in order to obtain a close performance estimation, with reduced sampling costs. If we ignore the network latency to storage compared to the disk access latency, the contribution of a cache hit in any level of cache to overall application performance is roughly the same. We use the most commonly deployed or proposed cache replacement policy, LRU, in deriving our cache performance model.

Because of the cache hierarchy *inclusiveness* property in a cache hierarchy using the LRU replacement policy at all levels, if an application is given a certain cache quota q_i at a level of cache i, any cache quotas q_j given at any lower level of cache j, will be mostly wasteful.

Hence, in an LRU cache hierarchy, only the maximum size quota given at any level of cache really matters for approximating the hit/miss ratio; therefore, we approximate the miss ratio of a two level cache, consisting of a buffer pool (c) and a storage cache (s) by the following formula:

$$\mathcal{M}(\rho_c, \rho_s) \approx \mathcal{M}(max[\rho_c, \rho_s]). \quad (4)$$

More details of the derivation of this formula, and the architecture of the storage system can be found in our previous work [9].

3.2.4 General Purpose Model Templates

Ensemble contains the following general purpose model templates that can be used to express analyst beliefs and build new model templates.

Gray-box Inverse Exponential Model (G-INV). Inverse_Exponential is a pre-defined mathematical relationship in Ensemble, defined as follows:

$$\hat{y}_{\alpha,\beta}(x) = \frac{\alpha}{x^{\beta}}. \quad (5)$$

In this formula, the parameters α and β are to be curve-fitted by Ensemble. Both the CPU-BOUND and the DISK-BOUND analytical models we described above are special forms of G-INV models.

Gray-box Region Model (G-RGN). The analyst believes that while the performance models of applications are complex in general, they are simple within a small range of configurations, i.e., constant, linear, or polynomial. Hence, we can model the performance using simple curve fitting within a region (i.e., a subset of configurations). While any function can be provided, for this model template, the analyst specifies the use of the average function for Ensemble to fit the samples in each region.

Black-box SVM Regression Model (B-SVM). Ensemble uses a black-box model template to cover all scenarios where no model is known, or to refine areas where other models do not provide sufficient accuracy. In this case study, Ensemble uses a well-known machine learning algorithm: *Support Vector Machine regression* [23] as its default, fully automated, black-box model template. SVM estimates the performance for configuration settings we have not actuated, through interpolation between a given set of sample points. SVM is shown to scale well for highly-dimensional, non-linear data. Radial basis functions (G-RBFs) are used as kernel functions.

Black-box Constant Model (B-CNST). This is a very simple model which uses a simple average relation which returns the average value of all training samples to predict performance. The predicted latencies are the same for all configurations. In contrast, G-RGN uses average function for each region, and hence the prediction values are usually different in different regions.

3.3 Direct Sampling Guidelines (Clues)

The analyst can directly specify *clues* indicating areas of the search space which can be pruned when sampling, because they are: already known, known to be noisy, or known to be invalid configurations.

For instance, from our experience, whenever the disk quanta given to an application is too small, e.g., less than 32 ms (about 0.1 fraction of the total disk quanta) in our storage server the disk latency shows unacceptably large variation. This is due to the lack of sufficient disk access time for the I/O burst. As an example, a clue for Ensemble to avoid sampling configurations with small disk quanta is shown in Listing 5.

```
1 CLUE Prune_Small_Quanta
2 ACTION Prune
3 CONDITION (a) {
4       a.name = 'disk_quanta' and
5       a.value <= 0.1
6 }
```

Listing 5. Prune Small Quanta Clue

3.4 Ensemble Design

We assume that a number of *model templates* have been provided for Ensemble to learn/validate within a configuration space C. We focus on describing how Ensemble trains and ranks performance models for C. For the purposes of training and ranking models per regions of applicability, Ensemble automatically divides the whole configuration space into multiple regions, controlled by a configurable parameter $\mathcal{D}$, which defines the number of divisions along each dimension. This results in dividing the configuration space into $\mathcal{D}^N$ regions, where N is the number of resource dimensions. Based on the contexts defined in the *model templates*, each model template to be used on the whole configuration space C or on a set of regions within C.

Algorithm 1 shows the learning process in Ensemble. The outcome is an overall model for C called an *ensemble* model. The algorithm uses performance samples gathered at runtime to refine each approximate model, and it ranks the models by accuracy, per region, within C. Our samples are gathered on the live system by actuation into the desired configuration and taking several measurements of the application's average memory access latency.

Algorithm 1. Iterative Algorithm to Build an Ensemble of Models for a Configuration Space C

1: Select a collection of model templates M
2: Divide configuration space evenly into l regions
3: Select v samples to construct test sample set $\mathcal{S}_v$.
4: Training set $\mathcal{S}_t = \varnothing$, $m = size(M)$
5: **repeat**
6: /* *Expand the training sample set* */
7: Add t new samples to the training sample set $\mathcal{S}_t$.
8: /* *Build* ensemble *of models* */
9: Partition the training set $\mathcal{S}_t$ into k subsets.
10: **for** $i = 1$ to k **do**
11: 1) Use ith subset as validation set $\mathcal{S}'_v$
12: 2) Use other $k-1$ subsets as training set $\mathcal{S}'_t$
13: **for** $j = 1$ to m **do**
14: 3) Train each base model M_j on $\mathcal{S}'_t$
15: 4) Test M_j on $\mathcal{S}'_v$
16: **end for**
17: **end for**
18: Derive rank per region from cross validation results
19: Build an *ensemble* from the rank
20: Test the *ensemble* of models on $\mathcal{S}_v$
21: **until** stop conditions are satisfied

Ensemble gathers a training sample set for C; the samples are gathered using user specified sampling methods (e.g., random sampling, greedy sampling, etc.) from the configuration space. The modeling refinement in our algorithm occurs *iteratively*, by adding new samples to the training set, until a stop condition is met.

In each iteration, we use the samples in the training set to i) build or refine, and rank the template-based approximate models and ii) evaluate the accuracy of our overall *ensemble* model for the whole configuration space C.

For this purpose, we further partition the training set into two sets: a *build set* for refining the template-based approximate models and a *validation set* for ranking them. Once the models are built or refined based on the *build set* within an iteration, we test their prediction accuracy using the *validation set*.

In more detail, we use a standard machine learning technique, called k-fold cross validation, for our model training and ranking process. The training sample set is partitioned into k subsets. Of these subsets, a single subset is taken as the *validation set*, and the remaining $k-1$ subsets are used as the *build set*. The cross-validation process is repeated k times, i.e., k folds, with each of k subsets of the training sample used exactly once as the validation data.

Ensemble ranks the models per region, based on their cross-validation results, and keeps the ranking results into a region table. The best ranked model in each region is selected to predict the performance for this region. Finally, in each iteration, we test our *ensemble* of models on a testing set, and compute its average relative error rate. If the test results satisfy the stop conditions, *Ensemble* is ready to be used for prediction on C; otherwise, the iterative training process continues. The stop conditions can be defined as an error rate threshold, a training time limit, or their combination.

4 Experimental Results

In this section, we first describe the benchmarks and platform we use in our evaluation. We use three standard industry benchmarks on our server platform *Akash* [9]. Akash is inspired by a cloud service, Amazon Relational Database Service [2], and consists of a storage hierarchy and allows to allocate buffer pool, storage cache and disk bandwidth to applications. Then, we present the results of modeling memory access latency with different workloads and show the benefits of using semantic guidance. Next, we evaluate the performance model generated by Ensemble in a resource allocation scenario as one of its use cases. Finally, we present an extension of Ensemble for heterogeneous environments and evaluate its performance in modeling a scientific application.

4.1 Testbed

We use two industry-standard benchmarks (TPC-W, TPC-C) and a OLTP-like workload OLTP-A to present our experience and evaluate *Ensemble*.

TPC-W [24] is a transactional web benchmark designed for evaluating e-commerce systems. We use the *browsing* workload, and scale up the workload; specifically, we created TPC-W^2 and TPC-W^{10} by running two and 10 TPC-W instances, respectively, in parallel, creating a database of 8 and 40 GB, respectively.

TPC-C [25] simulates a wholesale parts supplier that operates using a number of warehouse and sales districts. We use 128 warehouses, which gives a database of 32 GB.

OLTP-A is a workload generated using ORION (Oracle I/O Calibration Tool) [26], characterized by many random I/O accesses of 16 KB.

Our server platform is shown in Fig. 3. It consists of a database server running modified MySQL code and a virtual storage prototype, called *Akash*. We modify the MySQL/InnoDB to have a quanta based scheduler for CPU usage allocation, and modify its buffer pool implementation to support dynamic partitioning and resizing for each workload partition. The database server connects to Akash

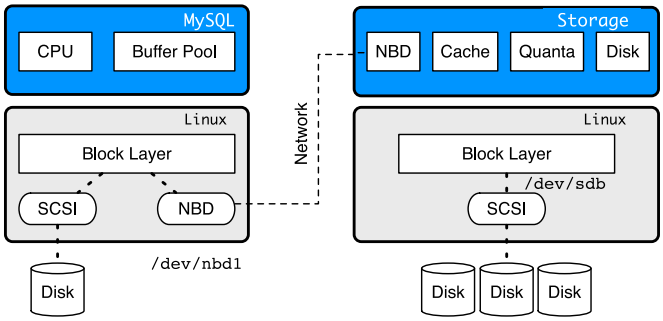


Fig. 3. Our server platform consists of a modified MySQL database server (left) and a virtual storage prototype Akash (right).

through the network, using Network Block Device (NBD) to mount a virtual volume as a NBD device (e.g., /dev/nbd1) which is used by MySQL as a raw disk partition, (e.g., /dev/raw/raw1).

Akash contains a storage cache which supports dynamic partitioning. We configure Akash to use 16 KB block size to match the MySQL/InnoDB block size. Also, Akash has a quanta based scheduler, which allocates the disk bandwidth to different workloads among several virtual volumes. The quanta-based scheduler partitions the bandwidth by allocating the resource in time quanta; within each quantum only one of the workloads obtains exclusive access to the underlying storage. Our platform thus provides strong isolation between workloads, hence we are able to measure the performance impact of resource allocations for every workload through dynamically setting its resource quanta. Using this platform, multiple applications can be hosted on the same database server, and share the underlying storage.

4.2 Modeling for Predicting Memory Access Latency

In this section, we evaluate Ensemble for modeling memory access latency (i.e. average buffer pool page access latency), for TPC-W[10] and TPC-C workloads, running on our server platform. In this example, Ensemble starts with the following model templates in its model template library: G-INV, A-STOR, G-RGN, B-SVM, B-CNST. We show how Ensemble refits these model templates from its model archive and validates them to model the memory latency of the two new workloads.

The configuration space of resources is designed as follows. We vary the size of the DBMS buffer pool from 128 to 960 MB with 15 settings, the storage cache size from 128 to 960 MB with eight settings for TPC-W[10], and 10 settings for TPC-C. We allocate the disk bandwidth in 32 ms quanta slices, and the disk quanta range from the minimum 32 ms quanta to the maximum 256 ms quanta with eight settings. The training time of exhaustive sampling is about 10 days for 960 configurations of TPC-W[10], and 13 days for 1,200 configurations of TPC-C. The size of the testing set is 10 percent of the original set size. The number of regions we divide on each resource dimension is 4, hence the whole configuration space is divided into 64 regions in our ensemble algorithm.

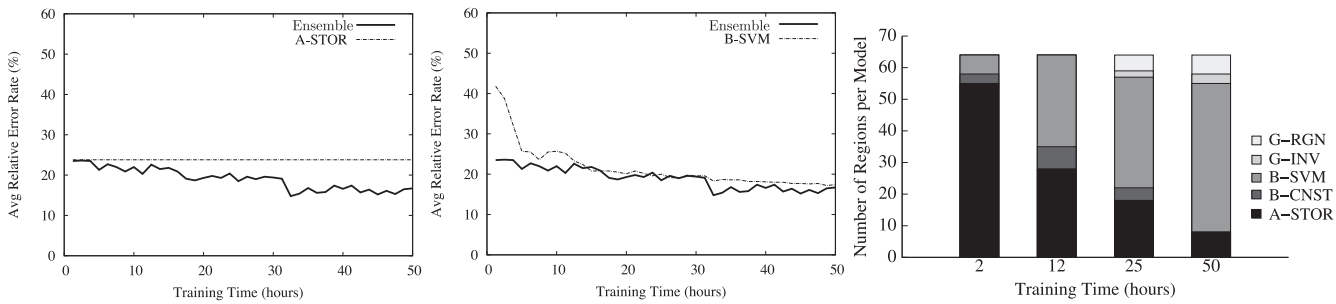
4.2.1 Modeling TPC-W[10] Memory Access Latency

Fig. 4 presents our results. On the x, we show the training time and the y shows the average relative error between the predicted and the measured performance for the testing set. For clarity, we show a trend curve comparison only between Ensemble and the best individual models. We also show how the accuracy of models changes over time in the ensemble model. With sufficient insight into the system, the A-STOR performs well with a constant error of 23 percent, shown in Fig. 4a. The black-box and gray-box models perform poorly initially but improve over time. Specifically, B-SVM initially has lower accuracy than the A-STOR model, but gradually improves to lower errors (below 20 percent). Other models, B-CNST, G-RGN and G-INV, while improving over time, perform poorly compared to A-STOR and B-SVM in most of the configuration space. As shown in Figs. 4a and 4b, *Ensemble* performs well matching the A-STOR model initially then incorporating the better predictions of B-SVM with more training time.

The results are further supported by Fig. 4c that shows that the A-STOR model is selected to predict performance for most regions initially, and then it contributes less to the *Ensemble* over time; the B-SVM model replaces the A-STOR over time. Specifically, after 50 hours of training time, the B-SVM model is selected as the best model in 50/64 regions of the configuration space. We can see again that no individual model always wins for all time deadlines and regions.

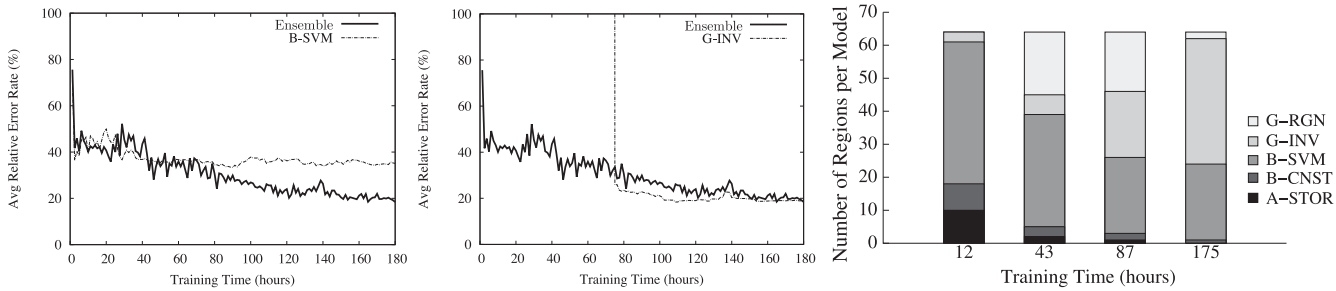
4.2.2 Modeling TPC-C Memory Access Latency

Fig. 5 shows our results. The black-box B-SVM model and the gray-box model G-INV contribute the most towards *Ensemble*. B-SVM performs with very high error (above 40 percent) for over 45 hours, then performs better with more training time until reaching about 30 percent error



(a) A-STOR (b) B-SVM (c) Contribution of Individual Models in Ensemble.

Fig. 4. **Performance prediction of memory access latency for TPC-W[10] workload.** Ensemble initially matches the A-STOR model, and then incorporates the better predictions of B-SVM model.



(a) B-SVM (b) G-INV (c) Contribution of Individual Models in Ensemble.

Fig. 5. **Performance prediction of memory access latency for TPC-C workload.** Ensemble initially relies on B-SVM model and later frequently selects gray-box models G-INV for prediction.

rate. G-INV takes a longer training time (more than 80 hours), and after sufficient training data, it is able to predict well. In fact, it has the lowest average error rates (about 20 percent) of any base model in the end. G-RGN has a similar trend as B-SVM, albeit with slightly higher average errors. On the other hand, the A-STOR and B-CNST models perform worse. Shown in Figs. 5a and 5b, *Ensemble* combines the positives of G-INV and B-SVM to achieve better performance. Specifically, it has a 20 percent prediction error (compared to 30 percent of B-SVM). In addition, by dynamically ranking the models, *Ensemble* performs better than G-INV for a long time (80 hours), and then matches the performance of the fully trained G-INV. The number of regions contributed by each base performance model is shown in Fig. 5c. It shows that initially the B-SVM is selected to predict performance for most of regions around 12 hours, and then it contributes less over time. While the G-INV is rarely selected initially, it becomes more frequently selected after a sufficiently long training time i.e., after 175 hours. The remaining regions are predicted using the G-RGN. We can see that the model composition is different from modeling TPC-W[10], and also that there is no single model that works best for all time deadlines.

4.3 Applying a Cache Pruning Clue

An example of expert knowledge that helped Ensemble prune the configuration space significantly is as follows. In this example, the analyst knows that the database buffer pool cache and storage cache use standard (uncoordinated) LRU replacement. Due to the cache inclusiveness property of LRU caches, she knows that any cache miss of a block from the buffer pool results in caching the block into both caches, i.e. the cache content is essentially replicated in both caches. Therefore, assuming the hit latency in either cache is about the same, whenever one of the caches is larger than the other cache, by at least some threshold value, the smaller cache is redundant, hence irrelevant. Based on this knowledge, the analyst can specify a *cache pruning clue* to use the same experimental sample points (e.g., a measured latency value) for a range of cache configurations that are roughly equivalent to its configuration by the above rule. For instance, using this clue, the latency value for the configuration [768 MB in buffer pool, 256 MB in storage cache] can be approximated using the latency value measured at the configuration point [768 MB in buffer pool, 384 MB in storage cache].

Table 1 shows the effects of applying this cache pruning clue on the modeling process. The time reduction refers to the experimental sampling time for modeling the memory access latency for the TPC-W[10] workload. If the threshold for pruning is 256 MB, the sampling time is reduced by 47.3 percent, from original 10 to five days, without any accuracy loss. If the threshold is 0 MB, the pruning causes a 6 percent increase in the average error rate, but at a dramatic time reduction (83.6 percent), i.e. sampling time reduces from 10 days to less than two days.

We apply the same clue for modeling TPC-C memory latency. The results are depicted in Table 2. In this experiment, if the threshold for pruning is 256 MB, the modeling time is reduced by 46.4 percent, from original 13 days to seven days. The modeling average error rate increases by 10 percent comparing with the error rate (20 percent) without applying the pruning clue. If the threshold is 0 MB, the pruning causes 14 percent increase in error rate, but reduces the sampling time to about one day, at 90.6 percent time reduction. For both cases, we can see that specifying sampling clues is a powerful method for sysadmin to express their domain knowledge and it can lead to dramatic sampling time reduction.

4.4 Ensemble use Case for Resource Allocation

In this section, we show two examples of using the *Ensemble* performance models that we built in Section 4.2, for allocating resources in our server platform. We assume that the goal of the resource allocation is to minimize the sum of the memory access latencies for all applications.

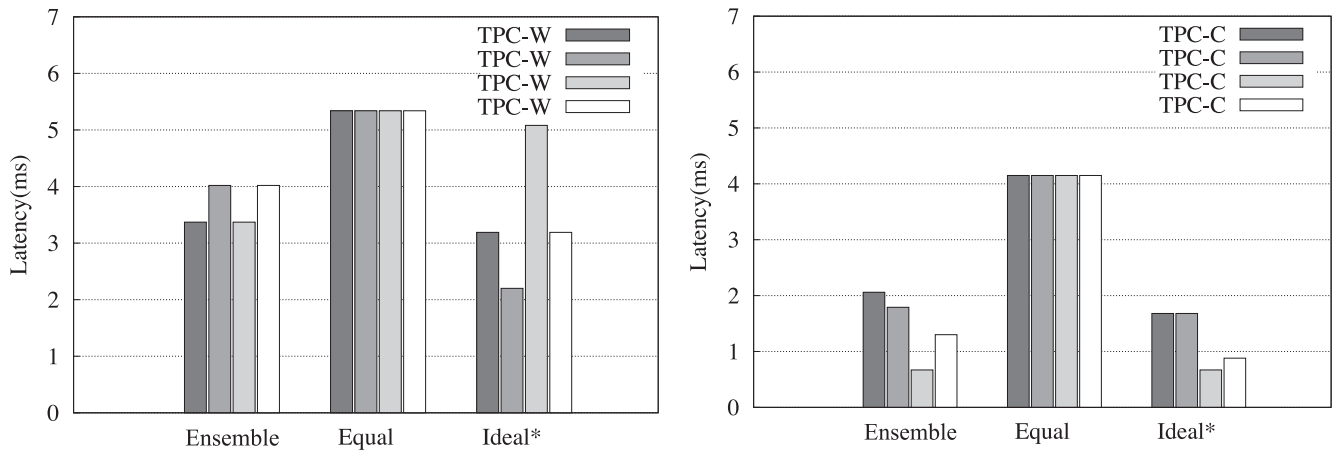
Based on the predicted latencies given by the performance models, Ensemble allocation scheme uses a hill

TABLE 1
Applying Cache Pruning Clue on TPC-W[10]

	Threshold (MB)	
	256	0
Time Reduction	47.3%	83.6%
Error Rate Increase	0%	6%

TABLE 2
Applying Cache Pruning Clue on TPC-C

	Threshold (MB)	
	256	0
Time Reduction	46.4%	90.6%
Error Rate Increase	10%	14%



(a) Running four TPC-W^{10} instances (b) Running four TPC-C instances

Fig. 6. Examples of resource allocation for running four identical instances of TPC-W.

climbing algorithm with random restarts to find the resource partitioning setting which gives the optimum performance. The hill climbing algorithm is an iterative search algorithm that moves towards the direction of decreasing combined latency value for all valid configurations at each iteration.

We compare the performance results (i.e. the sum of the latencies) using Ensemble resource allocation scheme that we described above with the following allocation schemes:

The Equal scheme assigns equal portions of the resources for each applications. The Ideal* scheme finds the configuration with the best overall latency by an exhaustive search through all resource configurations. This scheme is not feasible in practice, since it needs to exhaustively sample all configurations and may take weeks or even longer to finish.

We first run four instances of the TPC-W^{10} on our server platform, sharing the database and the underlying storage server. We set the size of both the buffer pool and the storage cache as 1 GB; and the disk bandwidth is partitioned among these instances. As shown in Fig. 6a, Equal has the worst performance with the overall latency 21.36 ms since it blindly partitions each resource by equal share, without considering the performance impact. In contrast, Ensemble uses the TPC-W^{10} performance models (with 16 hours of training) to guide its allocation decision. As a result, the overall latency is 14.78 ms by Ensemble scheme, very close to the Ideal* scheme (13.66 ms).

We observe similar trends for running four TPC-C instances on our server platform with the same resource constraints as the TPC-W^{10} allocation experiments. Shown in Fig. 6b, Equal performs worst with the highest overall latency 16.40ms; and the resource allocation scheme using Ensemble models for TPC-C(with 64 hours of training) achieves the total average latency 5.82 ms, almost as low as the Ideal* results (4.91 ms) .

The example use case shows that our Ensemble modeling framework can effectively support sysadmins to find near-optimum resource allocation.

4.5 Model Extension for Workload Mix Changes

Ensemble can reuse and extend trained historical models that are saved in Ensemble repository when the workload mix changes as we explain next.

We conduct a series of experiments to show the impact of reusing historical models when the workload mix changes, using OLTP-A benchmark. Specifically, initially a model trained for the workload with the write ratio of a has been saved into the Ensemble model repository, and we refer this workload as the historical or old workload. Then the write ratio of the data accesses decreases to b, and we refer this workload as the target or new workload, which we want to model. The Euclidean distance which reflects the similarity between the new and the old workload is denoted by variable $dist$, which equals $a - b$. We set the target error rate requirement for the modeling as 10 percent, and report the time of modeling for this new workload identified by the write ratio b.

Fig. 7 shows the results. The x-axis lists a series of tests with the same $dist$. The value of x-axis refers to the write ratio b in the new workload. y-axis shows the modeling time for the new workload with model reuse normalized to the time without reusing the historical model. That is, the modeling time is 1 if we train the new workload completely from the scratch. Each column corresponds one test. For example, first column of Fig. 7a shows that when we train a read-only workload from the old workload with 10 percent write ratio, and the normalized modeling time is reduced to about 0.32.

When the similarity distance $dist$ is 0.1, as shown in Fig. 7a, the modeling time is in the range of 0.2∼0.35, hence significantly shorter than in the case without model reuse. We further increase the distance $dist$ to 0.2. As shown in Fig. 7b, the modeling times for the new workload are longer than for previous tests with shorter distance (i.e., 0.1) due to the larger difference between the old and the new workloads, but overall the reductions in modeling time are still significant, in the range of 0.2∼0.7.

4.6 Model Extension for Heterogeneous Environments

Heterogeneous environments or significant variations in the infrastructure cause significant differences in the performance surface of the application. An example of such variations is in cases when the same application runs on different CPU and GPU architectures in the same cloud.

While historical models saved in the repository can share some fundamental characteristics with the new

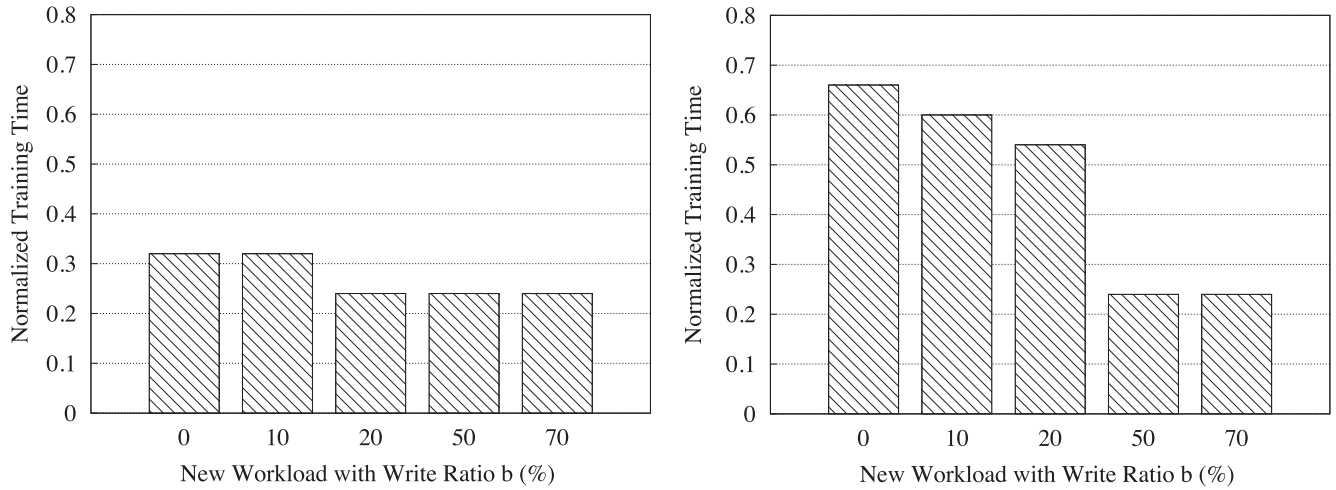


Fig. 7. Examples of model reuse when workload mix changes.

environment, reusing the best fitting model alone may be insufficient to model the new environment. Also, modeling all different environments separately is too expensive. In such situations, Ensemble uses a different strategy to reuse previously trained models. To build the model for a new environment, Ensemble creates a mapping model from a historical model to the performance surface of the new environment. This allows Ensemble to avoid rebuilding a new multi-dimensional model by building a single dimensional map instead. Our proposed method works as follows.

Starting from a historical performance model $\mu_1(\mathbf{c})$ that was built using n_1 samples, where **c** is the configuration vector, we intend to build the performance model of the new environment over the same configuration space ($\mu_2(\mathbf{c})$).

The model is a function of the multi-dimensional configuration parameter space (**c**) to the performance metric space (**p**). Then, to build the performance model of the new environment ($\mu_2(\mathbf{c})$), we build a mapping model (μ^*) of the performance surface of the historical environment ($\mathbf{p}_1$) to the performance surface of the new environment ($\mathbf{p}_2$). We use Gaussian Process models [27] to build the mapping model of the historical environment to the new environment iteratively as follows.

In the first iteration, we select two samples from n_1 samples of the historical environment with uniform distribution. The performance space of the historical environment (P_1) consists of all tuples of $<\overline{c}, p_1>$, where $\overline{c}$ is the configuration parameter vector and p_1 is the value of the target performance metric, e.g., application latency. This set can be read from the stored samples of the historical environment or can be generated on the fly using the historical model built and stored by Ensemble.

For each selected sample $<\overline{c}, p_1>$ from the performance space of the historical environment, we measure the performance of the new environment using the same $\overline{c}$ configuration parameters to build corresponding tuples of $<\overline{c}, p_2>$. By combining these two sample sets, we create P_2 a set of tuples of $<p_1, p_2>$. We use P_2 to build a mapping model of the performance of the new environment based on the performance of the historical environment using a Gaussian Process. The resulting Gaussian Process not only predicts performance value of the new environment, but also provides an estimation of the prediction error. We use this estimation to perform intelligent sampling rather than random, or user guided sampling.

At each subsequent iteration, we compute a sampling probability distribution which will be used to select the next sample from P_2. The goal is to choose the sample that maximizes reduction of model error. The available samples of the first model (P_1) may have a non-uniform distribution in the performance space of the first model, as it is the case for NPAIRS, our Neuroscience application (Fig. 8b). For this reason, to select sample(s) with uniform distribution, we compute the probability of selecting sample x_i in $<p_1, p_2>$ space (Fig. 8a) using equation (6):

$$prob(x_i) = \frac{1}{|bin_{x_i}|} \cdot \frac{\overline{MSE}_{bin_{x_i}}}{\sum_{j=1}^{H} \overline{MSE}_{bin_j}}, \tag{6}$$

where bin_{x_i} is the histogram bin of sample x_i and $\overline{MSE}_{bin_{x_i}}$ is the estimated mean squared error of the samples in bin_{x_i}.

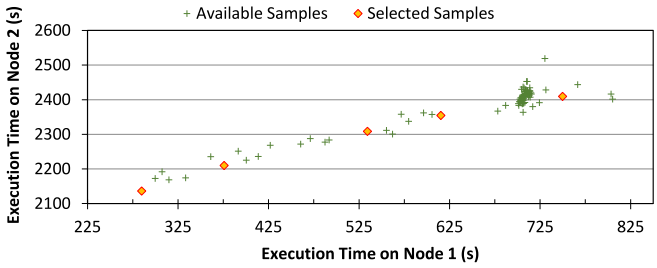
Using the sampling probability distribution computed in equation (6), the next sample p_1' is selected from the available pool of samples of the historical environment (P_1). Then, we use the configuration parameters associated with p_1', $\overline{c}_1'$, to measure the performance of the new environment p_2'. The new tuple $<p_1', p_2'>$ is added to the available sample set of the Gaussian Process model, and the model is rebuilt.

We continue the above process until a termination condition is reached. The termination condition can be a cap on the modeling time or the accuracy of the model. In our case, we use k-fold cross-validation technique to evaluate the accuracy of the model (μ^*). If the accuracy of the GP model does not increase significantly in two consecutive iterations, we stop the modeling process.

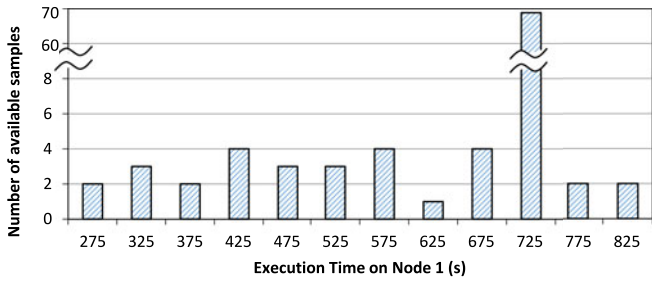
Finally, we compute the performance model of the new environment over the configuration space ($\mu_{new}(\overline{p})$) as equation (7):

$$\mu_2(\overline{c}) = \mu^*(\mu_1(\overline{c})), \tag{7}$$

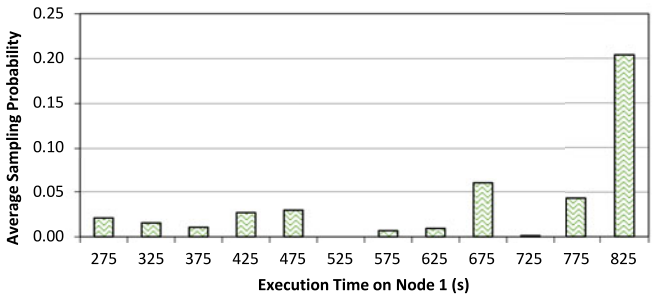
where μ^* is the mapping model built using the Gaussian Process and $\mu_1(\overline{p})$ is the historical model.



(a) Distribution of samples of Node1 and Node 2, i.e. execution time of NPAIRS, our Neuroscience application, on Node 1 and Node 2.



(b) Histogram of samples of Node 1.



(c) Average probability of sample bins of Node 1, computed using the estimation error of Gausian Process model based on the selected samples in (a).

Fig. 8. An example of distribution of available (potential) samples in the performance space of Node 1 and Node 2.

4.6.1 Evaluation

We evaluate our proposed method to build the performance model of NPAIRS running on an Intel Xeon E5-2650 CPU architecture (Node 2), starting from a stored Ensemble model of NPAIRS running on a NVIDIA Titan GPU architecture (Node 1). The performance surfaces of these environments are represented in Fig. 9.

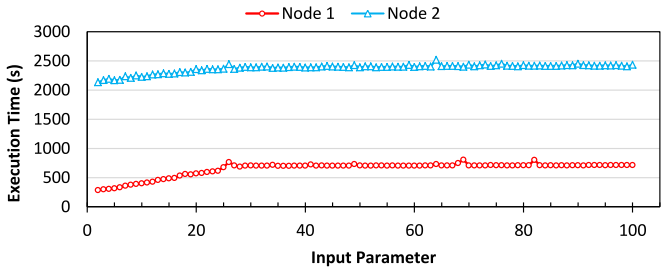


Fig. 9. Execution time of NPAIRS, our Neuroscience application case study on two different nodes, GPU-enabled Node 1 and CPU-enabled Node 2.

To show the effectiveness of our proposed method, we compare the quality and the sampling cost of building the performance model of Node 2 using the following methods:

- *Piecewise linear model*: The model selected and stored by Ensemble as the highest ranked model for the application running on Node 1.
- *GP Mapping Model*: The proposed Gaussian process mapping model.

For each method, we build the performance model of Node 2 using 2 to 30 samples as the training set. Then, for each model, we compute the Mean Squared Error of the test set, which contains all the available samples not present in the training set. We repeat this process 1,000 times and compare the average MSE of the methods.

The results depicted in Fig. 10 show that our proposed method reduces cost of sampling for the same error level threshold by a factor of two. In this case study the piecewise linear historical model stored by Ensemble eventually reaches a lower error level, although at a much higher cost. However, this is mainly due the geometry of the performance surface of NPAIRS, which is predominantly piecewise linear (Fig. 9).

5 Related Work

Existing techniques for predicting performance range from analytical models [20], [28], [29], [30] to black-box models based on machine learning algorithms [6], [31], [32], and gray-box models in between [33].

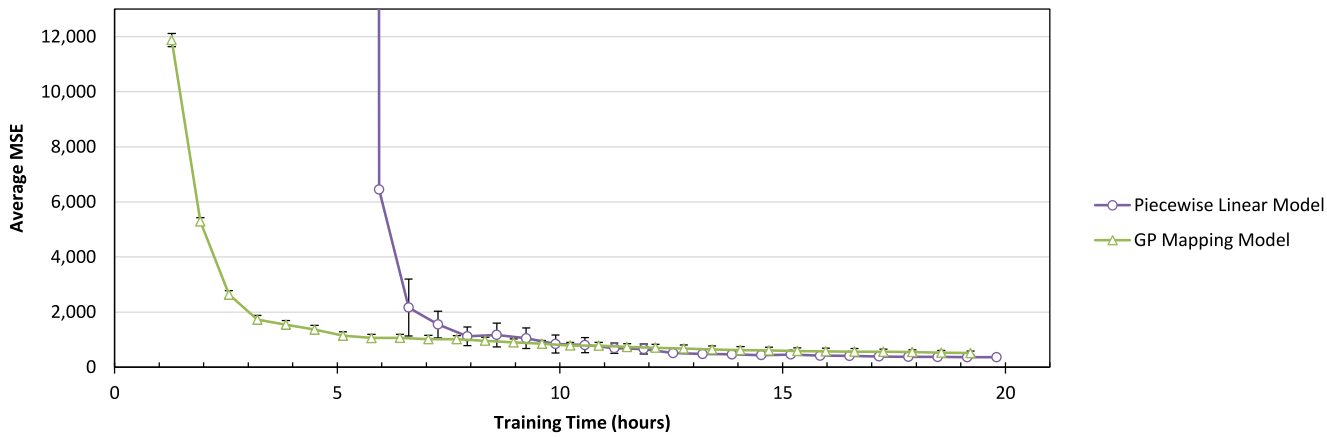


Fig. 10. Average MSE of the proposed Gaussian Process model versus piecewise linear model over 1,000 runs. To reach the same level of accuracy, the proposed Gaussian process model requires half of the training time (sampling time) of the piecewise linear model.

Building complete analytical models requires an in-depth understanding of the underlying system, which may not always be possible in complex heterogeneous cloud. As an example of advanced analytical models for specialized cases: Uysal et al. derive an analytical throughput model for modern disk arrays [30] and queuing models [20], [28] have been explored for CPU-bound web servers. Soror et al. [29] use query optimizer cost estimates to determine the initial allocation of CPU resources to virtual machines and to detect changes in the characteristics of the workloads.

With the complexity of modern systems, machine-learning based approaches have been explored to model these systems. Previous work either target providing a best model for some specific type of system or workload, or provide fully automated modeling methods. For example, Wang et al. use a machine learning model, CART, to predict performance for storage device [31]. Ganapathi et al. predict DBMS performance by using a machine learning algorithm [6] called KCCA. Zhang et al. [32] discuss how to use ensemble of a group of probability models for automated diagnosis of system performance problems. IRONModel [33] uses a hybrid decision-tree based machine learning model, called Z-CART, to predict the parameters for analytical models designed for their storage system [33]. iTuned [5] proposes an adaptive sampling method which automatically selects experimental samples guided by utility functions. Ghanbari et al. propose to use a query language for evaluating system behavior [34]. Recently, Herodotou and Babu propose to use a What-if Engine for MapReduce optimization [35]. They use relative black-box models to estimate cost statistics fields, and use analytical models to estimate data flow and cost fields. Recent papers [36], [37], [38] studied resource allocation problems using other prediction methods without semantic guidance used in our paper.

6 Conclusion

We designed, implemented, and deployed Ensemble, a novel runtime system for incremental, on-the-fly performance modeling of cloud data center applications. We observed that patterns of long running application classes are repeatable and hardware upgrades are sometimes localized to a few components. Based on our observations, our key idea is to provide an interactive high-level environment geared towards reuse and refinement of previously built models in new cloud configurations and workloads.

Ensemble partitions the configuration parameter space of a system, assigns a model template for each partition, and validates the model templates by intelligently sampling. In this way, Ensemble incrementally constructs an ensemble of models for the purposes of capacity planning, performance inquiry, resource allocation, performance anomaly detection and root cause analysis.

We showed that Ensemble can successfully validate, extend, and reuse existing simple models in order to approximate a memory latency model for a storage hierarchy and the performance model of a complex scientific application running in a heterogeneous cloud. Furthermore, we showed that these models are accurate enough for online resource management purposes.

In our future work, we will extend our model template library to accurately fit a larger variety of performance profiles. In addition, we will evaluate Ensemble on more complex applications with more elaborate performance profiles and higher dimensional configurations spaces. Also, we are planning to extend the capabilities of Ensemble and make it able to build models in noisy environments where sensory performance metrics may fluctuate due to external factors. Furthermore, we intend to study the effectiveness of applying Ensemble on building performance models of unknown applications based on a library of models of existing applications. One of the requirements is extending our library of models to incorporate a much larger variety of workload mixes and configurations.

Acknowledgments

Part of this work has been published in eighth International Workshop on Self-Managing Database Systems (SMDB 2013).

References

[1] Google App Engine [Online]. Available: http://developers.google.com/appengine, 2014.
[2] Amazon Relational Database Service (Amazon RDS) [Online]. Available: http://aws.amazon.com/rds/, 2014.
[3] Oprofile [Online]. Available: http://oprofile.sourceforge.net, 2014.
[4] Splunk [Online]. Available: http://www.splunk.com, 2014.
[5] S. Duan, V. Thummala, and S. Babu, "Tuning database configuration parameters with iTuned," *Proc. VLDB Endowment*, vol. 2, pp. 1246–1257, 2009.
[6] A. Ganapathi, H. A. Kuno, U. Dayal, J. L. Wiener, A. Fox, M. I. Jordan, and D. A. Patterson, "Predicting multiple metrics for queries: Better decisions enabled by machine learning," in *Proc. IEEE Int. Conf. Data Eng.*, 2009, pp. 592–603.
[7] A. Gulati, C. Kumar, I. Ahmad, and K. Kumar, "BASIL: Automated IO load balancing across storage devices," in *Proc. 8th USENIX Conf. File Storage Technol.*, 2010, pp. 169–182.
[8] G. Soundararajan, J. Chen, M. A. Sharaf, and C. Amza, "Dynamic partitioning of the cache hierarchy in shared data centers," *Proc. VLDB Endowment*, vol. 1, no. 1, pp. 635–646, 2008.
[9] G. Soundararajan, D. Lupei, S. Ghanbari, A. D. Popescu, J. Chen, and C. Amza, "Dynamic resource allocation for database servers running on virtual storage," in *Proc. 7th Conf. File Storage Technol.*, 2009, pp. 71–84.
[10] Q. Zhang, M. Zhani, R. Boutaba, and J. Hellerstein, "Dynamic heterogeneity-aware resource provisioning in the cloud," *IEEE Trans. Cloud Comput.*, vol. 2, no. 1, pp. 14–28, Jan. 2014.
[11] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. Gruber, "Bigtable: A distributed storage system for structured data," in *Proc. 7th Symp. Operating Syst. Des. Implementation*, Nov. 6–8, Seattle, WA, USA, 2006, pp. 205–218.
[12] L. Abraham, J. Allen, O. Barykin, V. R. Borkar, B. Chopra, C. Gerea, D. Merl, J. Metzler, D. Reiss, S. Subramanian, J. L. Wiener, and O. Zed, "Scuba: Diving into data at facebook," *Proc. VLDB Endowment*, vol. 6, no. 11, pp. 1057–1067, 2013.
[13] A. Gupta, F. Yang, J. Govig, A. Kirsch, K. Chan, K. Lai, S. Wu, S. G. Dhoot, A. R. Kumar, A. Agiwal, S. Bhansali, M. Hong, J. Cameron, M. Siddiqi, D. Jones, J. Shute, A. Gubarev, S. Venkataraman, and D. Agrawal, "Mesa: Geo-Replicated, near real-time, scalable data warehousing," *Proc. VLDB Endowment*, vol. 7, no. 12, pp. 1259–1270, 2014.
[14] B. Mozafari, C. Curino, and S. Madden, "Dbseer: Resource and performance prediction for building a next generation database cloud," in *Proc. CIDR*, 2013.
[15] J. Chen, G. Soundararajan, S. Ghanbari, and C. Amza, "Model ensemble tools for self-management in data centers," in *Proc. 8th Int. Workshop Self-Managing Database Syst.*, 2013, pp. 36–43.
[16] J. Chen, "Chorus: Model knowledge base for performance modeling in datacenters," Ph.D. dissertation, Univ. Toronto, Toronto, ON, Canada, 2011.

[17] S. C. Strother, J. Anderson, L. K. Hansen, U. Kjems, R. Kustra, J. Sidtis, S. Frutiger, S. Muley, S. LaConte, and D. Rottenberg, "The quantitative evaluation of functional neuroimaging experiments: The NPAIRS data analysis framework," *NeuroImage*, vol. 15, no. 4, pp. 747–771, 2002.
[18] S. Strother, A. Oder, R. Spring, and C. Grady, "The NPAIRS computational statistics framework for data analysis in neuroimaging," in *Proc. 19th Int. Conf. Comput. Statist.*, 2010, pp. 111–120.
[19] RUBiS [Online]. Available: http://rubis.ow2.org/, 2014.
[20] B. Urgaonkar, G. Pacifici, P. J. Shenoy, M. Spreitzer, and A. N. Tantawi, "An aonalytical model for multi-tier internet services and its applications," in *Proc. SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 2005, pp. 291–302.
[21] G. Soundararajan, M. Mihailescu, and C. Amza, "Context-Aware prefetching at the storage server," in *Proc. USENIX Annu. Techn. Conf.*, 2008, pp. 377–390.
[22] A. Verma, G. Dasgupta, T. K. Nayak, P. De, and R. Kothari, "Server workload analysis for power minimization using consolidation," in *Proc. Conf. USENIX Annu. Techn. Conf.*, 2009, p. 28.
[23] H. Drucker, C. J. C. Burges, L. Kaufman, A. J. Smola, and V. Vapnik, "Support vector regression machines," in *Proc. Adv. Neural Inf. Process. Syst.*, 1996, pp. 155–161.
[24] Transaction Processing Council [Online]. Available: http://www.tpc.org, 2014.
[25] F. Raab, "TPC-C - The standard benchmark for online transaction processing (OLTP)," in *The Benchmark Handbook*. Transaction Processing Council, 1993.
[26] ORION - Oracle I/O Calibration Tool [Online]. Available: http://www.oracle.com, 2014.
[27] P. Ranjan, R. Haynes, and R. Karsten, "A computationally stable approach to gaussian process interpolation of deterministic computer simulation data," *Technometrics*, vol. 53, no. 4, pp. 366–378, 2011.
[28] M. N. Bennani and D. A. Menascé, "Roesource allocation for autonomic data centers using analytic performance models," in *Proc. 2nd Int. Conf. Autom. Comput.*, 2005, pp. 229–240.
[29] A. A. Soror, U. F. Minhas, A. Aboulnaga, K. Salem, P. Kokosielis, and S. Kamath, "Automatic virtual machine configuration for database workloads," *ACM Trans. Database Syst.*, vol. 35, no. 1, pp. 953–966, 2010.
[30] M. Uysal, G. A. Alvarez, and A. Merchant, "A modular, analytical throughput model for modern disk arrays," in *Proc. 9th Int. Symp. Model., Anal. Simul. Comput. Telecommun. Syst.*, 2001, pp. 183–192.
[31] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger, "Storage device performance prediction with CART models," in *Proc. Joint Int. Conf. Meas. Model. Comput. Syst.*, 2004, pp. 412–413.
[32] S. Zhang, I. Cohen, M. Goldszmidt, J. Symons, and A. Fox, "Ensembles of models for automated diagnosis of system performance problems," in *Proc. Int. Conf. Dependable Syst. Netw.*, 2005, pp. 644–653.
[33] E. Thereska and G. R. Ganger, "Ironmodel: Robust performance models in the wild," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 2008, pp. 253–264.
[34] S. Ghanbari, G. Soundararajan, and C. Amza, "A query language and runtime tool for evaluating behavior of multi-tier servers," in *Proc. ACM SIGMETRICS Int. Conf. Meas. Model. Comput. Syst.*, 2010, pp. 131–142.
[35] H. Herodotou and S. Babu, "A what-if engine for cost-based MapReduce optimization," *IEEE Data Eng. Bull.*, vol. 36, no. 1, pp. 5–14, 2013.
[36] J. Duggan, Y. Chi, H. Hacigümüs, S. Zhu, and U. Çetintemel, "Packing light: Portable workload performance prediction for the cloud." in *Proc. ICDE Workshops*, 2013, pp. 258–265.
[37] J. Yin, X. Lu, H. Chen, X. Zhao, and N. N. Xiong, "System resource utilization analysis and prediction for cloud based applications under bursty workloads," *Inf. Sci.*, vol. 279, pp. 338–357, 2014.
[38] Q. Liang, J. Zhang, Y. Zhang, and J. Liang, "The placement method of resources and applications based on request prediction in cloud data center," *Inf. Sci.*, vol. 279, pp. 735–745, 2014.

Jin Chen received the MS and PhD degrees in computer science from the University of Toronto in 2005 and 2011, respectively. She received the BE and ME degrees in computer science and engineering from Beijing University of Aeronautics and Astronautics (BUAA) in 1998 and 2001. Her research interests include performance tuning and modeling, big data analytics, resource provisioning in large scale distributed systems.She has worked at Microsoft Research Asia, IBM Research China, University of Toronto, Neuro informatics group at Rotman Research Institute, BMO, Huawei as a system researcher or a senior software engineer. She is a member of the IEEE.

Gokul Soundararajan received the bachelor's of applied science degree in 2003 and the master's of applied science degree in 2005 from the University of Toronto, and the PhD degree from the University of Toronto in electrical and computer engineering in 2010. He is a member of technical staff at NetApp. He currently works on techniques to improve the efficiency and management of dynamic data centers. For his dissertation, he developed novel techniques to improve performance and the manageability of multitenant shared data centers. Other projects he has worked on include Griffin, a system to improve the lifetimes of SSDs, and a database provisioning system. He is a member of the IEEE.

Saeed Ghanbari received the MSc and PhD degrees in computer engineering from the University of Toronto in 2008 and 2013, respectively. Since then, he has been a member of technical staff at the Advance Technology Group in NetApp. His areas of research are on distributed systems, cloud-computing, and object storage. He is a member of the IEEE.

Francesco Iorio is a Distinguished Research Scientist and head of the Computational Science Group at Autodesk Research. His research focuses on software performance and scalability on multi-core and many-core systems, hybrid computing, accelerator systems, cloud computing and general design and exploitation of parallel algorithms on a variety of high performance platforms. Prior to Autodesk he was a Solution Architect for Next Generation Computing Systems at the IBM High Performance Computing Group in Dublin, where he was the lead solution architect for all Cell Broadband Engine processor (Cell/B.E.) related projects, with specific focus on accelerating financial, engineering and digital media workloads using hybrid high performance computing platforms. Previously, he was a Senior Real-Time Software Architect at Alias, where he was involved in design and development of high performance 3D computer graphics software applications. He received his MSc in Information Technology - Software Engineering from The University of Liverpool, UK.

Ali B. Hashemi received the MSc and PhD degrees in computer engineering from Amirkabir University of Technology (Tehran Polytechnic) in 2004 and 2011, respectively. He is a postdoctoral fellow at The Edward S. Rogers Sr. Department of Electrical & Computer Engineering, University of Toronto and an Intern researcher at Autodesk Research. His research interests include swarm intelligence, big data processing, and high-performance cloud computing. He is a member of the IEEE.

Cristiana Amza received the BS degree in computer engineering from Bucharest Polytechnic Institute in 1991, the MS and the PhD degrees in computer science from Rice University in 1997 and 2003, respectively. She is currently an associate professor with the Department of Electrical and Computer Engineering, University of Toronto. Her research interests are in the area of distributed and parallel systems, with an emphasis on designing, prototyping and experimentally evaluating novel algorithms and tools for self-managing, self-adaptive and self-healing behavior in data centers and Clouds. She joined the Department of Electrical and Computer Engineering at University of Toronto in October 2003 as an assistant professor and became an associate professor in July 2009. She is actively collaborating with several industry partners, including Intel, NetApp, Bell Canada, and IBM through IBM T.J. Watson, Almaden, and IBM Toronto Labs. She is a member of the IEEE.