Model Ensemble Tools for Self-Management in Data Centers

Jin Chen [#1], Gokul Soundararajan [*#2], Saeed Ghanbari [#3], Cristiana Amza [#4]

[#] *Department of Electrical and Computer Engineering, University of Toronto*
Toronto, Canada
[1] jinchen@cs.toronto.edu
[3] saeed@eecg.toronto.edu
[4] amza@eecg.toronto.edu

[*] *NetApp, USA*
[2] gokuls@netapp.com

***Abstract*— We introduce Ensemble, a runtime framework and associated tools for building query latency 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 database environments, such as data centers and Clouds. Ensemble combines simple, partially specified, lower-dimensionality models to provide good initial approximations for higher dimensionality, end-to-end query latency models.**

We perform an experimental evaluation on industry-standard applications running on a multi-tier dynamic content server. We show that the Ensemble on-the-fly modeling framework provides accurate, fast and flexible performance modelling by using partial, lower dimensionality models to approximate end-to-end query latency models.

I. Introduction

Large datacenters, e.g., Amazon Relational Database Service (RDS) datacenter [1], Google App Engine datacenter [2], are highly dynamic environments, co-hosting several applications sharing resources in a multi-tier server environment. Uncontrolled resource sharing between co-hosted applications often results in performance degradation problems, thus creating violations of service level agreements (SLAs) for service providers. Therefore, per-application *performance modeling* for *on the fly* adjustment of resource allocation to match per-application SLAs has recently become promising [3], [4], [5], [6], [7].

In this paper, we introduce Ensemble, a novel performance modeling framework for dynamic resource allocation or capacity planning in complex, service-hosting datacenter environments.

A *performance model* is a mathematical function that calculates an estimate of the application performance for a range of resource configurations. For example, Fig. 1 shows a performance model as a 3D surface for the online auctions application RUBiS. 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 buffer pool and storage. Similarly, the average query latency of an application varies on a 5D hyperplane, as a function of its resource quotas for CPU, buffer pool, storage cache, and storage bandwidth of that application.

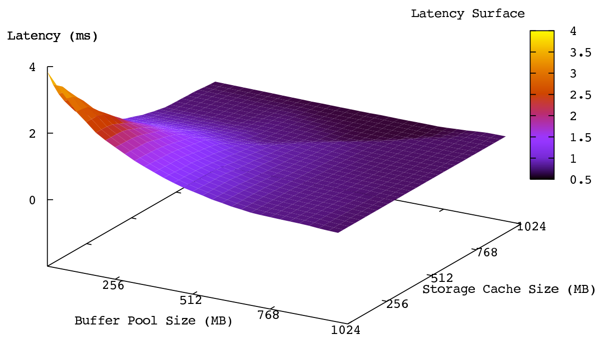


Fig. 1. RUBiS latency surface.

Automatic performance model building iterates through two steps: (i) gathering experimental samples, and (ii) modeling computation. Gathering experimental samples means actuating the experimental system into a given resource configuration, running a specific application workload on the live system (or equivalent) and measuring the application latency. Modeling computation involves mathematical interpolation for building the model on existing sampling data. While modeling computation is typically on the order of fractions of seconds, experimental sampling may take months for mapping out the entire resource 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 expensive even for a single configuration point.

For example, for N modeled resources, and M increments of sampling for each resource, an *application 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 warmup effect, experimental sampling takes around 15 minutes of measurements at *each* of the 1024 (32^2) points of the surface. The total sampling takes approximately 11 days.

In an enterprise environment, where 64GB storage caches are common, if we set sampling increments in 1GB units, total sampling would take 2 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 using analytical models that rely on sysadmin or analyst's semantic knowledge of the system and application [5], [7], [8]. However, these analytical models are precise only for restricted parts of the system, specific application workload mix or resource configurations. They are brittle to dynamic changes and require too much domain expertise.

In this paper, we propose to leverage automated black-box long-term learning of the system itself, coupled with administrator semantic awareness and expertise, wherever available to build an ensemble/mixture of models. The overall model thus built covers the end-to-end 5D query latency model with a satisfactory accuracy-time modeling trade-off in most situations.

We observe that calculating a number of potential models, or fitting known models over the collected sampling data is relatively fast compared to actuating and sampling. Moreover, disk operational laws and cache operational laws (e.g., cache LRU replacement) and disk access patterns [9] (e.g., sequential) are sometimes known beforehand, and can be taken into account to guide performance modelling.

Ensemble, our highly flexible performance modeling framework, uses these observations to provide lightweight sampling and modeling on the fly as well as accumulating a versioned model repository over time. At any given point in time, Ensemble provides performance predictions within the application's *current* operating zone, i.e., workload mix and resource availability. Ensemble can also automatically find configuration ranges where a model template fits its sampling data; it further automatically ranks models by accuracy per configuration region and selects the best model per region. In new situations, and wherever model templates are not available, Ensemble gathers experimental samples and uses black-box statistical regression to derive models.

II. Ensemble Framework

Long-term resource over-provisioning of resources in large data centers for all possible combinations of peak incidental loads is unacceptable in a dynamic environment, due to large cooling and power costs. Therefore, performance modeling of applications is desirable for automating resource allocation and what-if capacity planning. We further make the following assumptions about the application, environment, and system API that guide our modelling process.

A. Assumptions

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

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 query latency per application is assumed to be relatively stable during a *stable period.*

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 [7].

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 query 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 5D model. A similar function may apply for the cases where the application is CPU bound.

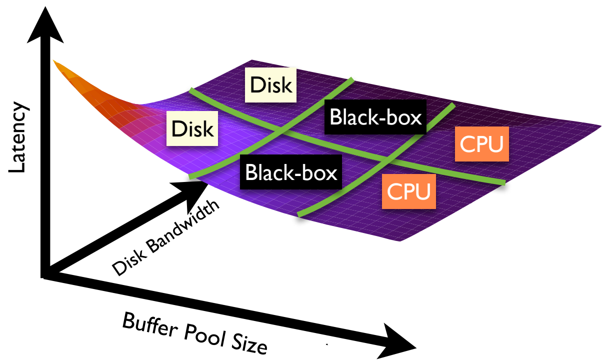


Fig. 2. An Example of Ensemble Learning.

Fig. 2 shows a simple interactive scenario visually for modeling a workload with Ensemble (although visual display would not be possible for higher dimensionality models).

The sysadmin doesn't 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, where the I/O is intensive. We refer the area where a

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

Listing 1. Syntax of Model Template

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 shown in the figure and also the operating region(s) of these models, if known. It is important to note that either Disk-bound or CPU-bound may have lower dimensionality than that of the configuration space of the overall model. Alternatively, the sysadmin can ask the Ensemble runtime system to automatically find the operation region for 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 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.

B. 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).

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*, Chorus 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.

```
TEMPLATE CPU_Bound
RELATION CPU_Inverse_Exponential(x,y) {
       x.name='cpu_quota' and
       y.name='query_latency'
}
CONTEXT (a) {
      a.name='memory_size' and a.value='*'
}
```

Listing 2. CPU_Bound Model Template

C. Basic CPU-Bound Query Latency Model (A-CPU)

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

$$\mathcal{R}_{query}(\rho_p, \rho_c, \rho_s, \rho_d) \quad = \quad \frac{\mathcal{R}_{\ query}(1)}{\rho_p} \tag{1}$$

where $\mathcal{R}_{query}(1)$ is the baseline query latency for an application, when all resources are fully allocated to that application, and ρ_p is the cpu quota allocated to the application.

Corresponding to this inverse exponential mathematical relation, a performance model, called A-CPU 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.

The relation is the Inverse_Exponential relation presented previously. 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.

D. Basic Disk-Bound Query Latency Model (A-Disk)

The analyst is aware that our 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 query 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 A-CPU model, as follows:

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

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 ∞).

This model is declared sensitive only to two parameters: the total effective memory size allocated to the application, and

```
TEMPLATE DISK
RELATION Inverse_Exponential(x,y) {
       x.name='disk_quota' and
       y.name='query_latency'
}
CONTEXT (a) {
      a.name='memory_size' and a.value='*'
      b.name='disk_quota' and b.value > 0.1
}
```

Listing 3. DISK Model Template

the disk quota allocated to the application. The configurations under which the model is considered effective exclude those with very low disk quantas. 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 32ms (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 "*".

E. Memory-Bound Query Latency Model (A-STOR-Q)

A-STOR-Q is an analytical model for the query latency designed for storage intensive workloads. Just like A-Disk, A-STOR-Q ignores the CPU time. However, A-STOR-Q is more sophisticated than A-Disk, because it models the access time to the memory hierarchy in more detail. In this model, our analyst leverages the query selectivity, which we obtain using our own statistics of the number of page accesses (i.e. $\mathcal{N}_{acc}$) to the database buffer pool.

The query latency model is:

$$\mathcal{L}_{query}(\rho_p, \rho_c, \rho_s, \rho_d) = N_{acc} * \mathcal{L}_{mem}(\rho_c, \rho_s, \rho_d) \qquad (3)$$

where $\rho_p, \rho_c, \rho_s, \rho_d$ is the CPU, buffer pool, storage cache and disk bandwidth quota allocated to the application; N_{acc} is the average number of page accesses made for each query in the workload; and $\mathcal{L}_{mem}$ is the average memory access latency for each page as derived in Equation 4, which we introduce below. This formula allows the analyst to express the miss rate relationship between the two caches as a function of the cache replacement policy, and is validated by the Ensemble run-time, just like any standard relation.

$$\begin{aligned}\mathcal{L}_{mem}(\rho_c, \rho_s, \rho_d) &= \underbrace{\mathcal{M}_c(\rho_c)\mathcal{H}_s(\rho_c, \rho_s)L_{net}}_{\text{I/Os satisfied by the storage cache}} \qquad (4)\\ &+ \underbrace{\mathcal{M}_c(\rho_c)\mathcal{M}_s(\rho_c, \rho_s)L_d(\rho_d)}_{\text{I/Os satisfied by the disk}}\end{aligned}$$

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, at much higher speed. Towards this goal, we use the experimental observation that the network latency is negligible compared to the disk access latency. This affords us the simplification that 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.

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, with $q_j < q_i$ will be mostly wasteful. This is because of the cache hierarchy *inclusiveness* property, where any cache miss from q_i will result in bringing the needed block into all lower levels of the cache hierarchy, before providing the requested block to cache i.

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:

$$\mathcal{M}(\rho_c, \rho_s) \approx \mathcal{M}(max[\rho_c, \rho_s]) \qquad (5)$$

F. General Purpose Model Templates

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

$$\hat{y}_{\alpha,\beta}(x) = \frac{\alpha}{x^{\beta}} \qquad (6)$$

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

2) Gray-box Region Model Template (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.

3) Black-box SVM Regression Model Template (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* [10] (B-SVM) 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.

4) Black-box Constant Model Template (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.

III. 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 will apply only to a part of the total regions in the total configuration space 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 query latency.

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.

Algorithm 1 Iterative algorithm to build an ensemble of models for a configuration space C.

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

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.

IV. Modeling Results

A. Testbed:

We use two industry-standard benchmarks (TPC-W, TPC-C) to present our experience and evaluate *Ensemble*. **TPC-W** [11] 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 2 and 10 TPC-W instances, respectively, in parallel, creating a database of 8 GB and 40 GB, respectively. **TPC-C** [12] simulates a wholesale parts supplier that operates using a number of warehouse and sales districts. We use 128 warehouses, which gives a database of 32GB.

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/Inn-

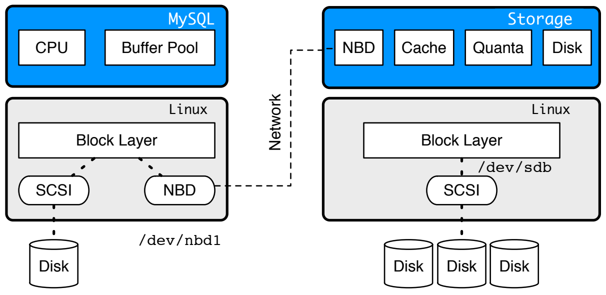


Fig. 3. Our server platform. It consists of a modified MySQL database server (shown in left) and a virtual storage prototype Akash (shown in right).

oDB 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 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, and has a quanta based scheduler, which allocates the disk bandwidth to different workloads among several virtual volumes. We configure Akash to use 16KB block size to match the MySQL/InnoDB block size. The quanta-based scheduler partitions the bandwidth by allocating the resource in time quantums; 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.

B. Building Models for Predicting Query Latency

In this section, we show how Ensemble refines preliminary models given as model templates and builds an ensemble of models with high accuracy for predicting the query latency of TPC-W^2, TPC-W^{10} and TPC-C, running on various configurations of our server platform. Ensemble validates and builds the ensemble of the following model templates. For modelling the query latency of TPC-W^2, we use just the basic Disk-bound (A-Disk) and CPU-bound (A-CPU) analytical model templates.

For modeling the other workloads, we replace the basic analytical model templates above with the equivalent, but more generic G-INV and we add all other model templates to the model knowledge base for Ensemble to fit, specifically: A-STOR-Q, G-RGN, B-SVM, and B-CNST.

1) Predicting TPC-W^2 *Query Latency :* We model the query latency for the TPC-W^2 workload within a region of the configuration space as follows: a range of DBMS buffer pools from 128M to 1024M, and a range of disk bandwidth quantas from 32ms to 256ms. The storage server does not have a storage cache in this experiment. The training time of exhaustive sampling is 30 hours for 120 configurations. This configuration space has three operating regions for TPC-W^2: (i) a CPU-intensive mode where the workload mostly fits in the buffer pool, (ii) an I/O-intensive mode, and (iii) a mixed mode influenced by both the allocation of the DBMS buffer pool and the disk bandwidth fraction.

We provide two partial analytical models (A-DISK and A-CPU) as our partial models without specifying/limiting the context for either of them within the configuration space above.

Fig. 4 presents our results. On the *x*-axis, we show the training time and on the *y*-axis we show the average relative error between the prediction and the measured performance for the testing set. The figure shows that each analytical model, on its own, performs well in its operating region, but poorly overall, thus resulting in high errors of 51% and 53%, for the A-CPU and A-DISK models, respectively. The analytical models cannot improve with more training samples (leading to the straight error lines in Fig. 4). On the other hand, the *ensemble* model performs better than both and improves with more training time. Specifically, Fig. 4 shows that after the 1 hour of training, the average error rate is about 30%, 20% after 2 hours, and 10% after 5 hours, at which time the *ensemble* model converges to the required accuracy. The benefit of the *ensemble* approach is that, by using performance samples, we can rank the two partial models for each region of the configuration space, and use the best to make the performance prediction. Furthermore, this experiment shows that, by leveraging just these two simple models as guidelines, the *ensemble* model can make accurate performance predictions even for the mixed mode area, thus substantially reducing the total training time compared to exhaustive sampling (from 30 to 5 hours of training).

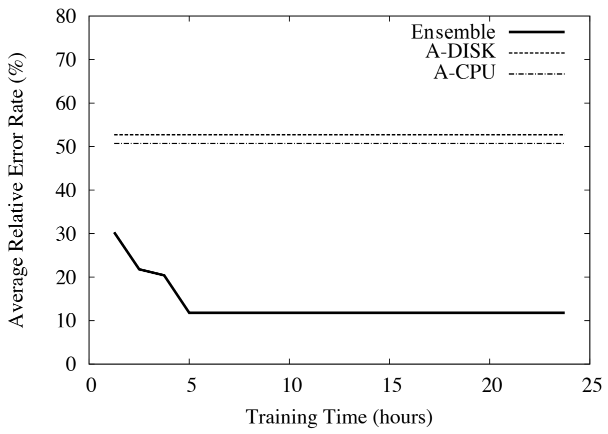
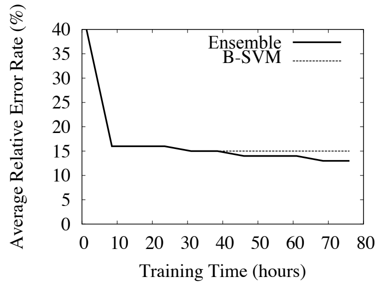
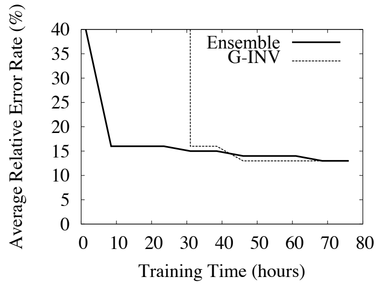


Fig. 4. Model TPC-W^2 workload. Ensemble leverages two simple, partial analytical models A-DISK and A-CPU for modeling the TPC-W^2 query latency. Error rates drop to 10% after 5 hours.

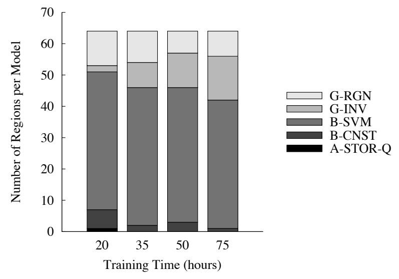
2) Predicting TPC-W^{10} *Query Latency :* We vary the size of the DBMS buffer pool from 128M to 960M with 15 settings, the storage cache size from 128M to 896M with 8 settings, and the disk bandwidth quanta from 32ms to 256ms with 8 settings. The training time of exhaustive sampling is about 10



(a) B-SVM



(b) G-INV



(c) 'Voices' of Individual Models in Ensemble.

Fig. 5. Performance prediction of query latency for TPC-W^{10} workload. Ensemble mainly matches the prediction of B-SVM model, and later incorporates the better predictions of G-INV model.

days for 960 configurations. The size of the testing set is 10% of the original set size. The number of regions used on each resource dimension is 4, hence the whole configuration space is divided into 64 regions.

Fig. 5 presents the results. The black-box B-SVM model performs well for this dataset. After about 10 hours of training, the average relative error quickly drops to about 15%. G-INV needs a longer training time (about 30 hours) to start predicting well. Ensemble mainly matches the prediction of the B-SVM model, and later it incorporates some better predictions from G-INV. On the other hand, the analytical model, A-STOR-Q, performs worse than B-SVM, with a high average error rate of 43%. The composition of Ensemble is shown in Fig. 5(c). The B-SVM model contributes most to Ensemble as it is ranked as the best model in more than 40/64 regions all the time. The G-INV model achieves slightly higher accuracy than B-SVM model after gathering sufficient training samples; as a result, it is ranked as the best model with more than 10 regions after 35 hours. The contribution of the analytical model, A-STOR-Q, to Ensemble is small, due to this model's inaccuracy. We can see the analytical model needs to be further improved in the modeling process of query latency.

Predicting TPC-C *Query Latency :* We vary the size of the DBMS buffer pool from 128M to 960M with 15 settings, the storage cache size from 128M to 896M with 10 settings, and the disk bandwidth quanta from 32ms to 256ms with 8 settings. The training time of exhaustive sampling is about 13 days for 1200 configurations. The size of the testing set is 10% of the original set size. The number of regions used on each resource dimension is 4, hence the whole configuration space is divided into 64 regions

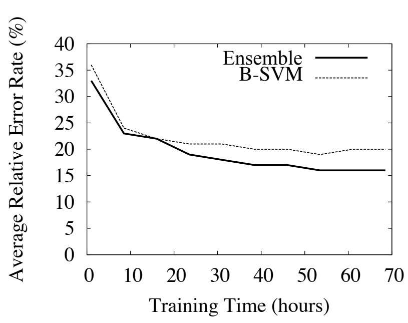
Fig. 6 shows the results. The black-box model B-SVM performs well initially, hence *Ensemble* matches its performance. After about 20 hours, *Ensemble* gradually outperforms B-SVM, with lower error rates. This is due to the fact that the G-RGN gray-box model starts to offer better predictions in many regions, and contributes more to *Ensemble*. From Fig. 6(c), the composition of *Ensemble* shows that it mostly selects the B-SVM model or the G-RGN model as the best ranked model per region for predictions. Through effectively combining the merits of both models, *Ensemble* outperforms each individual one. On the other hand, the more sophisticated analytical model, A-STOR-Q, does not perform well with a high average error rate of 77%, and is not selected into the composition of *Ensemble*. A secondary, but important point is that, from the Figures presenting our results for TPC-W^{10} and TPC-C, we can see that no individual model always wins all the time; hence it is crucial to dynamically validate and rank the models.

V. Related Work

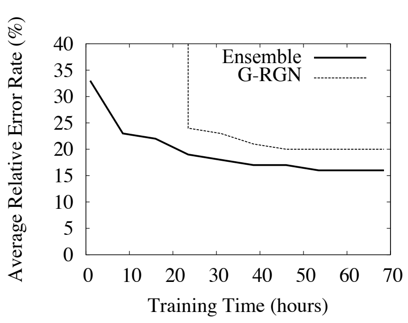
Existing techniques for predicting performance range from analytical models [13], [14], [8], [15], to black-box models based on machine learning algorithms [4], [16], [17]. There are also previous gray-box models in related studies [18]. Furthermore, Zhang et al. use a regression-based analytical model for capacity planning of multi-tier applications [19]. Compared to these existing systems, Ensemble seamlessly combines analytical performance models that have been derived for specialized systems with generic types of models.

Building complete analytical models requires an in-depth understanding of the underlying system, however, which may not always be possible in multi-tier server systems. As an example of advanced analytical models for specialized cases: Uysal et al. derive an analytical throughput model for modern disk arrays [15] and queuing models [13], [8] have been explored for CPU-bound web servers. Soror et al. [14] 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.

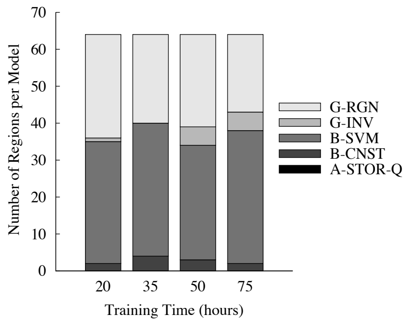
With the complexity of modern systems, machine-learning based approaches have been explored to model these systems. These previous works either target providing a best model for some specific type of system or workload, or provide fully automated modeling/sampling methods. For example, Wang et al. use a machine learning model, CART, to predict performance for storage device [16]. Ganapathi et al. predict DBMS performance by using a machine learning algorithm [4] called KCCA. Zhang et al. [17] discuss how to use ensemble of a



(a) B-SVM



(b) G-RGN



(c) 'Voices' of Individual Models in Ensemble.

Fig. 6. Performance prediction of query latency for TPC-C workload. Ensemble initially matches the prediction of B-SVM model, and soon incorporates the better predictions of G-RGN model.

group of probability models for automated diagnosis of system performance problems. IRONModel uses a hybrid decision-tree based machine learning model, called Z-CART, to predict the parameters for analytical models designed for their storage system [18]. iTuned [3] proposes an adaptive sampling method which automatically selects experimental samples guided by utility functions. While our approach has some similarities with these approaches, our Ensemble automatically leverages different types of models, each of which may work well in a different operating mode, to provide a hybrid model that can make accurate performance predictions in all regions.

VI. Conclusion

We design, implement and deploy Ensemble, a novel runtime system for incremental, on-the-fly performance modeling of datacenter applications. Ensemble validates model templates using monitoring data for the resources the model is sensitive to. In this way, Ensemble incrementally constructs an ensemble of models for the purposes of capacity planning, performance inquiry or resource allocation.

Through a set of preliminary studies, we show that Ensemble can successfully validate, extend and reuse existing simple models in order to approximate an end-to-end 5D query latency model. Furthermore, we show that a model for query-latency as a function of resource allocation can be built with around 40 hours of training, with an inaccuracy below 15% in a nearly black-box way, without any knowledge of query plans, database statistics, or details of the hardware environment.

References

[1] "Amazon Relational Database Service (Amazon RDS)," http://aws.amazon.com/rds/.

[2] "Google App Engine," https://developers.google.com/appengine/.

[3] S. Duan, V. Thummala, and S. Babu, "Tuning Database Configuration Parameters with iTuned," *Proceedings of the VLDB Endowment*, vol. 2, pp. 1246–1257, 2009.

[4] 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 *Proceedings of the 25th International Conference on Data Engineering (ICDE'09)*, 2009, pp. 592–603.

[5] A. Gulati, C. Kumar, I. Ahmad, and K. Kumar, "BASIL: Automated IO Load Balancing Across Storage Devices," in *Proceedings of the 8th USENIX Conference on File and Storage Technologies (FAST'10)*, 2010, pp. 169–182.

[6] G. Soundararajan, J. Chen, M. A. Sharaf, and C. Amza, "Dynamic Partitioning of the Cache Hierarchy in Shared Data Centers," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 635–646, 2008.

[7] 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 *Proceedings of the 7th USENIX Conference on File and Storage Technologies (FAST'09)*, 2009, pp. 71–84.

[8] 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 *Proceedings of the International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS'05)*, 2005, pp. 291–302.

[9] G. Soundararajan, M. Mihailescu, and C. Amza, "Context-Aware Prefetching at the Storage Server," in *Proceedings of the 2008 USENIX Annual Technical Conference (USENIX'08)*, 2008, pp. 377–390.

[10] H. Drucker, C. J. C. Burges, L. Kaufman, A. J. Smola, and V. Vapnik, "Support Vector Regression Machines," in *Proceedings of the Annual Conference on Neural Information Processing Systems (NIPS'96)*, 1996, pp. 155–161.

[11] "Transaction Processing Council," http://www.tpc.org.

[12] F. Raab, "TPC-C - The Standard Benchmark for Online transaction Processing (OLTP)," in *The Benchmark Handbook*. Transaction Processing Council, 1993.

[13] M. N. Bennani and D. A. Menascé, "Roesource Allocation for Autonomic Data Centers using Analytic Performance Models," in *Proceedings of the 2nd International Conference on Autonomic Computing (ICAC'05)*, 2005, pp. 229–240.

[14] 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, 2010.

[15] M. Uysal, G. A. Alvarez, and A. Merchant, in *Proceedings of the 9th International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'01)*.

[16] M. Wang, K. Au, A. Ailamaki, A. Brockwell, C. Faloutsos, and G. R. Ganger, "Storage device performance prediction with CART models," in *Proceedings of the International Conference on Measurements and Modeling of Computer Systems (SIGMETRICS'04)*, 2004, pp. 412–413.

[17] S. Zhang, I. Cohen, M. Goldszmidt, J. Symons, and A. Fox, "Ensembles of Models for Automated Diagnosis of System Performance Problems," in *International Conference on Dependable Systems and Networks (DSN'05)*, 2005, pp. 644–653.

[18] E. Thereska and G. R. Ganger, "Ironmodel: Robust Performance Models in the Wild," in *Proceedings of the International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS'08)*, 2008, pp. 253–264.

[19] Q. Zhang, L. Cherkasova, N. Mi, and E. Smirni, "A Regression-based Analytic Model for Capacity Planning of Multi-tier Applications," *Cluster Computing*, vol. 11, no. 3, pp. 197–211, 2008.