Adaptive Learning of Metric Correlations for Temperature-Aware Database Provisioning

Saeed Ghanbari, Gokul Soundararajan, Jin Chen†, Cristiana Amza
Department of Electrical and Computer Engineering
†Department of Computer Science
University of Toronto

Abstract

This paper introduces a transparent self-configuring architecture for automatic scaling with temperature awareness in the database tier of a dynamic content web server. We use a unified approach to achieving the joint objectives of performance, efficient resource usage and avoiding temperature hot-spots in a replicated database cluster.

The key novelty in our approach is a lightweight on-line learning method for fast adaptations to bottleneck situations. Our approach is based on deriving a lightweight performance model of the replicated database cluster on the fly. The system trains its own model based on perceived correlations between various system and application metrics and the query latency for the application. The model adjusts itself dynamically to changes in the application workload mix. We use our performance model for query latency prediction and determining the number of database replicas necessary to meet the incoming load. We adapt by adding the necessary replicas, pro-actively in anticipation of a bottleneck situation and we remove them automatically in underload. Finally, the system adjusts its query scheduling algorithm dynamically in order to avoid temperature hot-spots within the replicated database cluster.

We investigate our transparent database provisioning mechanism in the database tier using the TPC-W industry-standard e-commerce benchmark. We demonstrate that our technique provides quality of service in terms of both performance and avoiding hot-spot machines under different load scenarios. We further show that our method is robust to dynamic changes in the workload mix of the application.

1 Introduction

This paper introduces a new pro-active resource allocation technique based on on-line learning for the database back-end of dynamic content web sites. Dynamic content servers are complex systems which commonly use a three-tier architecture (see Figure 1) that consists of a front-end web server tier, an application server tier that implements the business logic, and a back-end database tier that stores and queries the dynamic content of the site. The cooling and power costs of gross hardware over-provisioning for each application's estimated peak load are making efficient resource usage crucial for large sites hosting several applications.

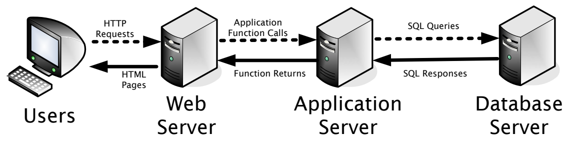


Figure 1. Three-tier Architecture

Fully-transparent, provisioning solutions [5, 12, 23] have been recently introduced to address this problem. These solutions add servers to an application's allocation based on queuing models [5, 26], utility models [23, 25], or marketplace approaches [9]. These existing approaches have demonstrated good accuracy in simulations or experimentally. On the other hand, they typically do not model multiple database replicas as part of their solution search space [5, 14, 15, 25, 26]. The challenge lies in accurately modeling the behavior of a replicated database backend tier. Many factors, such as database concurrency control, the wide range of query execution times typical in e-commerce workloads, load balancing policy, caching effects and replica consistency maintenance algorithm may influence performance. The derivation of a classic analytical model taking into account even a subset of these factors can be time consuming.

Our previous work [7, 22] in this area uses either a simple reactive scheme [22] to provision database replicas to workloads or requires extensive off-line training [7] of system states for each representative workload mix in order to accurately adapt on-line.

In this paper, we first introduce a new database replica provisioning scheme which adapts on-line with minimal off-line analysis. As before, the main goal is to maintain the application query latency within a predefined Service Level Agreement (SLA) value. We then add as a secondary goal avoiding temperature hot-spots for the cluster nodes in the database server tier. Previous work on modeling temperature profiles [16] has shown that avoiding hot-spots by maintaining the maximum temperature of the nodes in the cluster as low as possible translates into cost savings for cooling. Hence, we enhance our dynamic provisioning technique with temperature awareness for avoiding hot-spots and we study the temperature effects of various temperature-aware and temperature-unaware techniques.

The key novel feature of our database provisioning approach is on-line learning of correlations between system states and the response time under those states. A system state is defined as the load on the database back-end as reflected by measured system metrics, and the configuration of the database back-end, i.e., the number of database replicas for the respective application. Intuitively, a correlation is an automatically determined function that allows us to extrapolate future behavior from previously seen system metrics and configurations and their corresponding response times.

Not all database system metrics are well correlated with response time. We use minimal off-line analysis to guide the selection of the most promising correlations that the system should automatically determine on-line. The system then accumulates and memorizes a sample set of recent metric and database configuration points and their respective response times that are representative of the learned correlation function. We use *Support Vector Machine* (SVM) [10] regression for determining the correlations between measured metrics for a range of database configurations on one hand and response time on the other hand. SVM is a powerful machine learning approach which can capture non-linear feature correlations. The correlations are determined on-line based on the accumulated set of sample points.

The system then tracks the error of the correlation function and either adds new points or replaces existing points in the sample set on the fly, when the measured error is beyond an acceptable margin. In this way, the system automatically adapts the on-line correlation function to reflect new load and database configuration situations that it has not previously seen. The correlation function also evolves on-line in response to dynamic changes in the application and system, such as a workload mix shift or a change in database hardware. The learned correlations form a lightweight system model that evolves on-line and allows us to predict response time as a function of measured system metrics and predicted load. The model also allows us to determine the appropriate database tier configuration necessary for a predicted level of load. Based on this prediction, we can add database replicas to an application in advance of load spikes and we can remove a number of database replicas from the allocation of a particular application when they are no longer necessary.

Finally, we enhance our database provisioning algorithm with load shifting based on temperature-awareness. The algorithm uses unloaded replicas opportunistically to improve the temperature profile of the database cluster. Our baseline provisioning algorithm adds replicas in *advance* of predicted load spikes and uses conservative estimates for removal of replicas. Hence, newly added replicas will normally not be used for some time before and after the load spike actually happens. Our algorithm uses this time as a window of opportunity to shift the load from the highest temperature replica currently used to the coldest replica allocated to the application. Spreading the load across all machines in the application's allocation is another simple method to reduce the load, hence temperature of hot-spot machines. The more conservative the allocation of additional replicas, the more opportunities for temperature-aware load shifting or load spread. However, the load spread method can make a difference in per-node temperature only if the per-machine load after the load spread is significantly lower than the original. In contrast, we can significantly lower the temperature of the highest temperature machine with only one additional replica through load shifting.

We evaluate pro-active versus reactive provisioning schemes using the shopping and browsing mix workloads of the TPC-W e-commerce benchmark [24], an industry-standard benchmark that models an online book store. We drive the web site with a varying load function. We perform experiments where the workload mix dynamically changes between browsing and shopping. Finally, we instrument our cluster with temperature monitoring and we compare the temperature profile of temperature-aware schemes based on load shifting and uniform load spread with the baseline temperature-unaware provisioning scheme.

Our results are as follows:

1. By triggering adaptations early, our novel pro-active scheme with on-line learning avoids most SLA violations under a variety of load scenarios, including sharp load increases.

2. Our correlation tracking is shown to be robust to on-line workload mix changes.

3. Both of our temperature-aware schemes reduce the maximum temperature in the cluster compared to the temperature-unaware scheme, thus avoiding hot-spots. At the same time, the temperature-aware schemes have similar performance to the temperature-unaware pro-active scheme. The load shifting scheme uses fewer resources than the uniform load spread technique, hence it is preferable in constrained resource scenarios.

The rest of this paper is organized as follows. We first introduce the necessary background related to the SVM learning algorithm in Section 2. We then introduce our system architecture and give a brief overview of our dynamic replication environment in Section 3. We introduce our proactive approach with and without temperature awareness in Section 4. We describe a baseline reactive provisioning approach used for comparison in the experiments in Section 5. Section 6 describes our experimental testbed and benchmark. Section 7 illustrates the experimental results of our pro-active adaptation approaches with and without temperature awareness under different workload patterns and dynamically varying workload mixes. We compare our work to related work in Section 8. Finally, we conclude the paper in Section 9.

2 Background

We use Support Vector Machines (SVM) [10] as our regression algorithm for on-line metric correlation determination. In SVM the goal is to find a smooth function $f(x)$ that has a small deviation (ε) from the targets y_i for all training data. In other words, errors less than ε are tolerated while larger deviations are not acceptable. Suppose we are given training data consisting of l samples, $\{(X_1,y_1),(X_2,y_2),\ldots,(X_l,y_l)\}$ where X denotes the space of the input samples. Then, the estimated function $f(X)$ takes the form:

$$f(X) = \sum_{i=1}^{l} \alpha_i y_i K(X_i, X) \qquad (1)$$

Each training point X_i is associated with a variable α_i that represents the strength with which the training point is embedded in the final function. The points which lie closest to the hyper plane, denoting $f(X)$, are called the *support vectors*. $K(X_i,X)$ is a kernel function which maps the input into a high dimensional space, called feature space, where linear support vector regression is applied. Typical kernel functions include linear kernel, RBF kernel, sigmoid kernel, and polynomial kernel. The training of SVMs is a convex optimization problem solved using quadratic programming.

SVM works well for highly dimensional and non-linear data. Unlike instance-based learning algorithms, such as k-nearest-neighbors (KNN), SVM automatically chooses the appropriate number of support vectors, thus avoiding the need to store all training samples. Finally, SVM trains fast compared to other machine learning algorithms, such as KNN, and converges to an unique solution.

3 System Architecture

Our replicated database cluster architecture uses a set of schedulers, one per application, interposed between the application and the database tiers. The scheduler tier distributes incoming requests to a cluster of database replicas. The application server tier views the scheduler as a virtual database while the database tier views the scheduler as the application server. Upon receiving a query from the application server the scheduler sends the query using a scalable and strongly consistent read-one, write-all replication scheme, called *Conflict-Aware* replication [2] to the replica set allocated to that application. The replica set is chosen by a resource manager that makes the replica allocation decisions across the different applications.

Since database allocations to applications can vary dynamically, each scheduler keeps track of the current *database set* allocated to its application. We further distinguish the database set into an application's read and write replica sets, called *read set* and *write set* respectively. The read set is the set of machine replicas from which an application reads. Likewise, the write set of an application is the set of replicas that are maintained fully up to date with the writes of the application through our underlying replication scheme. We load balance the read queries among the replicas in the read set. For a particular application, the *read set* and *write set* may be different. However, the read set of an application is always a subset of its write set. The scheduler is also in charge of bringing a new replica up to date by a process we call *data migration* during which all missing updates are applied on that replica.

Complementing the scheduler is a resource manager that manages resource allocations of different applications. The schedulers keep track of average query performance metrics per sampling interval and communicate performance monitoring information periodically to the resource manager. The resource manager, based on its global knowledge of each application's SLA requirements and their perceived performance makes database allocation decisions for all applications. The decisions are communicated to the respective application schedulers, which act accordingly by including or excluding databases from their database read and/or write sets for their corresponding application.

4 Temperature-Aware Proactive Provisioning with On-Line Learning

In the following, we first give an overview of our proactive replica allocation to meet performance requirements. Then we discuss our dynamic performance model generation, load prediction and provisioning policies in more detail. Finally, we introduce our temperature-aware enhancements to the pro-active provisioning technique.

4.1 Overview

Our proactive approach employs SVM regression to dynamically generate a performance model correlating system metrics to the average query response time. The obtained model is used to predict the number of databases that can execute queries without SLA violation. We combine the model with a load prediction scheme to predict load at several points in the future, and proactively allocate databases in anticipation of load increase. In addition, using the model enables the resource manager to avoid over-provisioning by de-allocating replicas to each workload based on the predicted number of database servers it needs. Our proactive scheme uses three modules: Tracking, Load prediction and Provisioning. The Tracking module is responsible for learning the correlation of system states and latency. The Load prediction module predicts the number of incoming queries, which is further used by the Provisioning module to determine the best machine allocation for the application. The Provisioning module is an algorithm incorporated into the resource manager which translates the predicted number of databases needed for each application over a sequence of future time intervals to allocation or deallocation commands to the scheduler. In the following subsections we describe each module in more detail.

4.2 Tracking Module for Dynamic Performance Model Generation

Our performance model generation is based on a novel correlation detection scheme, Online Adaptive Tracking (OAT), for correlating load intensity and query latency.

OAT generates a dynamic performance model of the system online in the form of a function $y_{pred} = f(\vec{x})$, where $\vec{x} = (c_1, ..., c_p, n)$ is a vector of measured system metrics c_i's, called load features, characterizing the load on the system, n is the number of database replicas allocated to the application, and y_{pred} is the predicted query latency.

The model learns the correlation function $f(.)$ dynamically and thus can adapt to changing application or environment conditions. Having the correlation function $f(.)$ enables us to plan for adding or removing replicas based on predicted values for load features. In order to reduce overhead on the system, we preselect load features that are well correlated to response time through off-line analysis.

In a database system, multiple factors can influence the average query response time. We consider nine features of our database tier as measured during a sampling interval that reflect load characteristics as follows: Average query throughput (QueryThput), Average number of active connections (ActiveConn) used to deliver one or more queries from the application servers as detected by the scheduler, Rate of incoming Queries (Query) at the scheduler, Rate of incoming Read Queries (RD), Rate of incoming Write Queries (RW), and Lock ratio (Lock), i.e., number of locks held versus total queries.

While discarding the least promising correlations could theoretically be done on-line, we currently perform this filtering off-line. We use two filtering methods, correlation coefficient [21] and cross-validation [11]. Correlation coefficient is a well-known statistical technique for discovering highly correlated dimensions. We validate correlations between measured metrics and query latency through cross-validation.

Our Online Adaptive Tracking (OAT) algorithm uses a small set of samples of observed system metrics, their database configurations and the resulting query latency for obtaining the correlation function. For this purpose we use Support Vector Machine Regression (SVM).

The OAT accumulates its sample set L adaptively. Sample set $L = \{[\vec{x}(i), y(i)] \in R^{p+1} \times R, i = 1, ..., l\}$ consists of l pairs $(\vec{x}_1, y_1), (\vec{x}_2, y_2), ..., (\vec{x}_l, y_l)$. $f_L(\vec{x})$ represents a function fitted over the set L using SVM.

The sample set is formed dynamically by keeping the best descriptive samples. A sample is descriptive when it closely represents the relationship of y and $\vec{x}$ for regression at the current time. To maintain the dynamic set highly descriptive, hence keep the accuracy of prediction high, we update the sample set upon encountering a prediction error over an acceptable threshold. Specifically, if the absolute value of the difference of $y_{pred} = f_W(\vec{x}_{cur})$ of current system state $\vec{x}_{cur}$, and the measured metric y_{actual} is more than a predefined threshold α, OAT picks the closest sample in the sample set, and replaces it with a linear combination of the old and new sample by the following formula:

$$(y_{new}, \vec{x}_{new}) \leftarrow \mu(y_{closest}, \vec{x}_{closest}) + (1-\mu)(y_{actual}, \vec{x}_{cur}) \tag{2}$$

Closeness is defined as $l_p - norm$ of difference of x_{cur} and x part of samples: $(y_{closest}, \vec{x}_{closest}) = \arg\min_{(y,\vec{x}) \in L}(l_p - norm(\vec{x} - \vec{x}_{cur}))$, μ is a non-negative number between 0 and 1. The replacement of the old sample with the new sample as shown in the equation 2 evolves the sample set so that recent changes in the relation of $\vec{x}$ and y are refelected in the sample set. On the other hand, this evolution is done gradually, such that an errant measurement cannot significantly influence the sample set.

4.3 Load Prediction Module

We use a standard linear extrapolation technique for future load prediction based on the history of load evolution. Studies for analysis and characterization of web workloads [13, 3, 4] show that a standard linear extrapolation is generally sufficient for prediction.

Performance of linear predictors can be sensitive to the prediction window. A long prediction window tolerates transient noises (i.e., small load spikes) at the cost of a potentially delayed reaction to sudden, but significant load change. In contrast, a small prediction window responds to sudden change quickly at the cost of being highly influenced by transient noises. To overcome this issue, we use a combined set of linear regressions with different window sizes to predict the load. Multiple window sizes allow us to capture the dynamics of load better than a fixed window size. We use the least mean square method [11] for linear regression with different window sizes, and select the result of the predictor which has the largest correlation coefficient. This statistic is a measure of how well a straight line describes the data.

4.4 Provisioning Module

Once the performance model and predicted load is known, we can decide the best combination of database replica allocations at various time intervals in the future. Our algorithm determines the minimum number of databases to be allocated to an application subject to keeping the predicted query response time below the SLA.

The best allocation at time t, $\Theta_t = \{N(p,t) | \forall p \in P\}$, where $N(p,t)$ denotes allocation to application p at time t, and P denotes the set of applications, is determined through solving the following combinatorial optimization problem:

$min \sum_{\forall p \in P}(N(p,t) + \lambda_p s_p)$
$w.r.t$
$\forall p : f_p(\vec{x}_{pred}(t), N(p,t)) \leq SLA_{app} + s_p, s_p \geq 0$
$\sum_{\forall p} N(p,t) \leq M$
$\forall p : N(p,t) \geq 1$

where M is the maximum number of replicas, $\vec{x}_{pred}(t)$ is the predicted load at time t, $f_p(.)$ is the response time function for application p provided by OAT, and λ_p is a positive real constant which determines the penalty of SLA violation for application p if the demand exceeds the resources available and a violation is unavoidable. The objective function is to i) minimize the number of replicas allocated to each application and ii) minimize the penalty of SLA violation, subject to three constraints. The first constraint is to keep the average response time for each application below a predefined SLA. The second constraint is that the number of replicas allocated to each application should not exceed the total resources available. The last constraint ensures that each application has at least one replica in its allocation at any time. If the number of replicas needed for all applications exceeds the number of available database servers, the slack variables s_p's in the objective function and the first constraint enforce the least possible SLA violation. We use a greedy algorithm to solve the optimization problem.

The Provisioning module is responsible for instructing the scheduler of each application when and how to add or remove databases. This task is not trivial because of the following reasons. The first concern is the long adaptation delay. Adding a database replica to an application's allocation is a time-consuming operation. The database state of the new replica is typically stale and must be brought up-to-date via the application of missing updates before it can be used. In addition, the buffer cache at the replica needs to be warm before the replica can be used effectively. An associated problem is the period of instability that occurs during adaptations, when system metrics may not reflect typical system behavior. For example, when adding a replica, the system metrics might show abnormal values due to the load imbalance between the old and newly added replicas. We have shown that a pro-active approach that disregards the needed period of system stabilization after adaptation induces system oscillations between rapidly adding and removing replicas [7].

The provisioning module uses estimates of the adaptation delay time for adding database replicas to trigger allocation sufficiently in advance of predicted need. Furthermore, it uses a simple finite state machine to determine "cycles", i.e., cases where databases are added then removed within a period of time shorter than the estimated adaptation delay, including update and ramp-up time. It then filters out these reconfiguration actions of the database tier as unnecessary. Furthermore, in order to avoid oscillation in replica allocation, the provisioning module allows a replica removal for the application's allocation only when the system is stable. For stable system states, an increase in response time typically correlates to an increase in throughput during the same interval. Instability is detected when variation of throughput reversely correlates with variation of response time, e.g., decrease in throughput coincides with increase in response time.

4.5 Temperature Aware Scheduling Enhancements

We enhance our proactive provisioning technique with temperature aware scheduling. The objective of temperature-aware scheduling is to minimize the temperature of the hottest machine within the set of machines allocated to each application. Previous work has shown that avoiding hot-spots in a cluster results in a temperature profile conducive to cooling cost savings. Per-node temperature is generally a function of CPU utilization, but is also dependent on the physical location of the node and how it is exposed to air conditioning flows. For example, in our cluster configuration, the air conditioning unit is located at the floor level. Thus, machines at the top of a cluster rack receive warmer inlet air than the ones closer to the floor. Finally, the CPU temperature on any given machine is not a linear function of CPU utilization and typically reaches its peak faster than the CPU saturation point.

Our proactive resource management allocates a number of database servers to each application in anticipation of a load spike and removes them with a conservative delay after the spike. Thus, before and after the load-spike actually happens, only a subset of the allocated machines are required to meet the query latency SLA. We use this opportunity to improve the temperature profile of the cluster by periodically shifting the read query load of hot machines onto unloaded cool machines. During periods of stable load, when all machines are used fully, we can also allocate a small number of additional machines for the application for this purpose. The technique requires that temperature monitoring of the cluster machines is in place and the temperature of each machines is periodically polled.

The CPU temperature on each node increases mainly on account of heavyweight read query execution rather than due to execution of write queries. The temperature-aware algorithm keeps all replicas up-to-date by sending write queries to all replicas allocated to that application. However, the application's read-set is changed periodically by excluding the hottest machine and incorporating the coolest unused machine.

Sending read-queries to all replicas allocated to an application is an alternative. The idea is to spread load uniformly on all machines allocated to an application in such a way to have lower CPU utilization for all replicas, hence lower the maximum temperature. The all-replica scheme has the advantage that it does not need instrumentation of the cluster to measure per-node CPU temperature. However, this scheme may need a significant number of additional machines in order to spread the load to the point where the reduced CPU utilization makes a difference in terms of temperature reductions. Since each additional machine used consumes energy, the temperature-aware load-shift scheme is expected to provide similar temperature reductions at better resource usage.

5 Baseline Reactive Provisioning Approach Used for Comparison

We use a baseline reactive scheme [22] for comparison. In the reactive scheme, we add or remove a single replica to/from an application allocation based on two latency thresholds: a `HighSLAThreshold`, and a `LowSLAThreshold`, respectively.

A resource manager monitors the average latency of each application and compares it to the two given thresholds. For this purpose, it uses a smoothened latency average computed as an exponentially weighted mean of the form $WL = \alpha \times L + (1-\alpha) \times WL$, where L is the current query latency. The larger the value of the α parameter, the more responsive the average to the current latency. If the average latency over the past sampling interval for a particular workload exceeds the `HighSLAThreshold`, hence an SLA violation is imminent, the resource manager adds a database to that application's allocation. Thereafter, to account for replica addition delay, the resource manager stops making allocation decisions based on sampling query latency until the completion of the replica addition process.

If the average latency is below a `LowSLAThreshold`, the resource manager triggers a replica removal. The removal path is conservative and involves a tentative remove state before the replica is finally removed from an application's allocation. The allocation algorithm enters the tentative remove state when the average latency is below the low threshold. In the tentative remove state, a replica continues to be updated, but is not used for load balancing read queries for that workload. This two-step process avoids system instability by ensuring that the application is indeed in underload.

6 Experimental Setup

To evaluate our system, we use the TPC-W e-commerce benchmark and the same hardware for all machines in our cluster running the client emulator, the web servers, the schedulers and the database engines. All machines contain Dual 3.00Ghz Intel Xeon processors with HyperThreading enabled with 2GB of RAM and a 224GB hard drives. All nodes are connected through 1Gbps Ethernet LAN. All the machines use the Ubuntu Linux operating system. We run the TPC-W benchmark using three popular open source software packages: the Apache 1.3 web server [1] with PHP 4.0 [20] implementing the business logic and the MySQL 4.0 database server with InnoDB engine [17] to store the data.

All experimental numbers are obtained running an implementation of our dynamic content server on a cluster of 8 to 16 database server machines. We use a number of web server machines sufficient for the web server stage not to be the bottleneck. The largest number of web server machines used for any experiment is 6. We use one scheduler and one resource manager.

We configure the reactive provisioning algorithm with a *HighSLAThreshold* of 600ms and a *LowSLAThreshold* of 200ms. In our proactive algorithm, we set the soft SLA threshold to 600$\pm$100ms. The SLA threshold was chosen conservatively to guarantee an end-to-end latency at the client of at most 1 second for the TPC-W workload. We use a latency sampling interval of 10 seconds for the scheduler. Unless otherwise stated, for all experiments, the maximum size of the adaptive sample set that OAT keeps is 50 samples.

6.1 TPC-W E-Commerce Benchmark

The TPC-W benchmark from the Transaction Processing Council [24] is a transactional web benchmark designed for evaluating e-commerce systems. Several interactions are used to simulate the activity of a retail store such as *Amazon.com*. The database size is determined by the number of items in the inventory and the size of the customer population. We use 100K items and 2.8 million customers which results in a database of about 4GB.

The inventory images, totaling 1.8GB, are resident on the web server. We implemented the 14 different interactions specified in the TPC-W benchmark specification. The complexity of the interactions varies widely, with interactions taking between 20 ms and 1 second on an unloaded machine. Read-only interactions consist mostly of complex read queries in auto-commit mode. These queries are up to 30 times more heavyweight than read-write transactions. We use the TPC-W shopping and browsing workload mixes with 20% and 5% writes, respectively.

6.2 Client Emulator

To induce load on the system, we have implemented a session emulator for the TPC-W benchmark. For each customer session, the client emulator opens a persistent HTTP connection to the web server and closes it at the end of the session. Each emulated client waits for a certain think time before initiating the next interaction. The next interaction is determined by a state transition matrix that specifies the probability of going from one interaction to another. The session time and think time are generated from a random distribution with a given mean. For each experiment, we use a load function according to which we vary the number of clients over time. However, the number of active clients at any given point in time may be different from the actual load function value at that time, due to the random distribution of per-client think time and session length. For ease of representing load functions, in our experiments, we plot the input load function normalized to a baseline load.

6.3 Temperature Monitoring

We use the embedded Baseboard Management Controller (BMC) using the Intelligent Platform Management Interface (IPMI) interface. The BMC monitors many hardware components such as the CPU, the fans, and the motherboard. In addition, the BMC logs server fault events, alert administrators of server faults, and enables basic remote operations. We use the `ipmitool`[1] to access the BMC and record the temperature of the CPUs. Every minute, we poll the BMC sensors and record the results in a MySQL database.

[1]`http://ipmitool.sourceforge.net/`

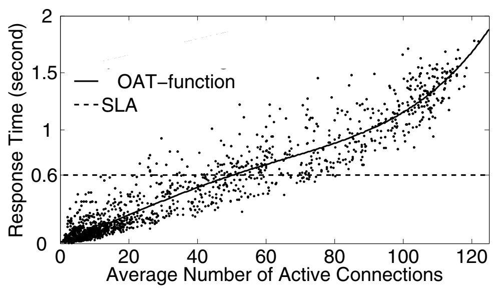


Figure 2. Correlation of number of active connections and response time

7 Experimental Results

In this section, we first show preliminary off-line experiments for load feature selection. Next, we show the benefit of our proactive provisioning over the reactive approach. Then, we show that our proactive provisioning scheme with on-line learning can adapt quickly to a change in workload mix. Finally, we show that our proactive algorithm augmented with temperature awareness allows us to decrease the maximum temperature in the cluster while still meeting the SLA.

7.1 Preliminary Experiments

As shown in Table 1, the *average number of active connections* has a strong correlation with the *average query response time*. We also verified this observation through cross-validation by running OAT with each of the features alone, and some combinations of them. Figure 2 shows a set of sample points and the correlation between the average number of active connections and query response time for one database server. We can see that the average number of active connections has a close to linear correlation with the average query response time when the response time is less than the SLA. As the response time increases further, the correlation becomes non-linear. We select the average number of active connections as the load feature for correlation tracking purposes.

7.2 Proactive versus Reactive Provisioning

In this section, we compare our proactive provisioning scheme with a baseline reactive scheme. In this experiment, we test our system with the TPC-W browsing mix and a maximum of 600 clients for 25 minutes. As shown in Figure 3(a), we use a varied load function with patterns ranging from small steps (between 9-14 minute marks) to sharp drops (at the 20 minute mark) and sudden spikes (at the 14 minute mark). Figure 3(b) shows the response time of the

Table 1. Correlation Coefficients for Various Metrics and Query Latency

METRICS	Latency	ActiveConn	QueryThput	Query	RD	RW	Lock
Latency	1.0000	0.9012	0.4335	0.4290	0.4419	0.3630	0.1943
ActiveConn	0.9012	1.0000	0.7163	0.7170	0.7247	0.6434	0.4115
QueryThput	0.4335	0.7163	1.0000	0.9993	0.9919	0.9466	0.6546
Query	0.4290	0.7170	0.9993	1.0000	0.9932	0.9456	0.6557
RD	0.4419	0.7247	0.9919	0.9932	1.0000	0.9016	0.5970
RW	0.3630	0.6434	0.9466	0.9456	0.9016	1.0000	0.7467
Lock	0.1943	0.4115	0.6546	0.6557	0.5970	0.7467	1.0000

reactive and proactive provisioning schemes. Figure 3(c) and Figure 3(d) show the number of replicas allocated by reactive and proactive provisioning algorithms respectively. The first 7 minute interval of the experiment is designed to allow the OAT to observe system states with different numbers of replicas in the database configuration and train itself. During this time, the algorithm falls back on the reactive approach whenever the SLA is violated. As we can see from the associated latency graph, we register significant SLA violations during this initial part. The subsequent 14 minutes of the experiment measure the effectiveness of the proactive scheme based on on-line learning compared to the reactive provisioning approach.

The figures show that the proactive scheme handles the load spikes well, meeting the SLA within the margin of error for the tracking algorithm and without overprovisioning replicas. In contrast, the reactive scheme registers substantial SLA violations during load spikes, and it allocates more replicas as the load ramps. As we can see, the proactive scheme is more precise in terms of replica allocation than the reactive scheme, by allocating 4 replicas and keeping the SLA violations within the (10%) allowed threshold bound as compared to 7 replicas allocated by the reactive scheme to handle the same load spike.

7.3 Adapting to Change of Workload Mix

We show the ability of OAT to accurately predict the average response time by adapting on-line to a change in the workload mix. Our system is tested with a sinusoid workload pattern with interleaved shopping and browsing workload mixes as shown in Figure 4(a). Our experiment lasts for 90 minutes. We vary the load from 1 to 400 clients for the browsing mix and between 1 and 600 clients for the shopping mix. Figure 4(c) shows that OAT can adapt well dynamically to the change of workload mix.

Figure 4(c) shows the response time. As before, in the early stages (first 10 minutes) of the experiment, the OAT exhibits many SLA violations while learning the correlation function. During this period, the OAT is collecting the sample set and modifying the sample set for a workload mix change. After two iterations of each mix, we see that OAT can allocate replicas in advance of need to avoid SLA violation regardless of the workload mix. Overall, during this experiment, the average absolute error for OAT prediction of query latency based on the correlation function was 37 ms. Moreover, the query latency was below the SLA for 96% of the time intervals during the whole experiment.

7.4 Evaluation of Temperature Aware Scheduling

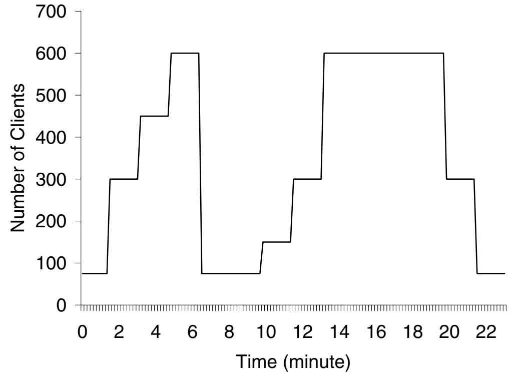
In this section we compare our temperature-aware scheduling schemes, load shifting and uniform load spread, with the proactive scheme without temperature awareness. We conduct two experiments.

In the first scenario, shown in Figure 5, we compare the maximum temperature registered in the cluster for all schemes at the same resource usage. To obtain the same resource usage for all schemes (in this case 7 machines in the read set of the application and 11 machines in the write set), we disable replica removal after a load spike and perform the different scheduling algorithms on the same number of replicas. This scenario can be considered representative of a data center in light load, where no other applications currently compete for resources.

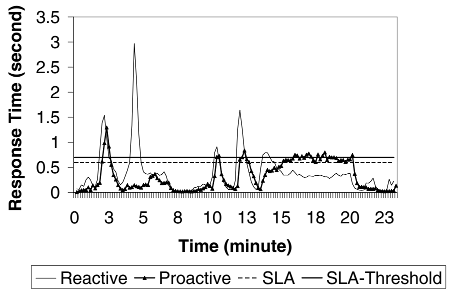
Figure 5(a) shows the maximum temperature of the 11 replicas during this experiment. The all-replica load spread scheme has the best temperature profile followed by load shifting temperature-aware scheduling, while the temperature-unaware scheme has the highest temperature. All schemes have almost ideal SLA compliance in this case, with very low query latency as shown in Figure 5(b).

In the second scenario, we compare the three schemes with respect to all our objectives: meeting the SLA, using resources efficiently and lowering the maximum temperature across the cluster in a resource constrained scenario.

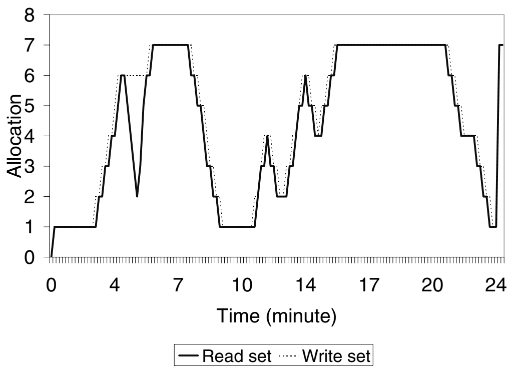
In this experiment we induce a flat load function with 500 emulated clients running the TPC-W browsing mix. The scheduler is constrained to use only 7 replicas with 5 of them in the read set of the application. As shown



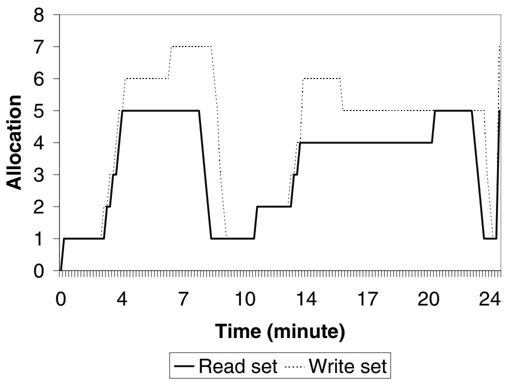
(a) Load Function



(b) Average latency

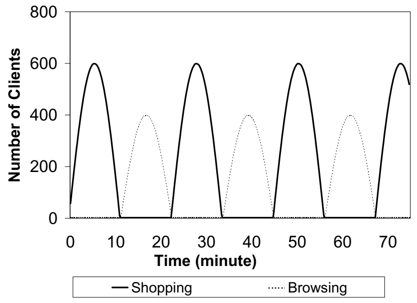


(c) Replica Allocation (*Reactive*)

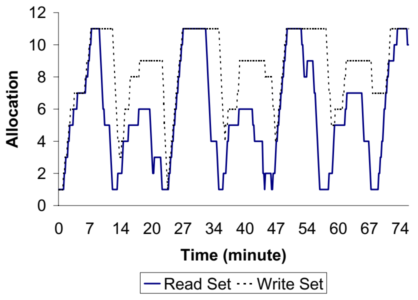


(d) Replica Allocation (*Proactive*)

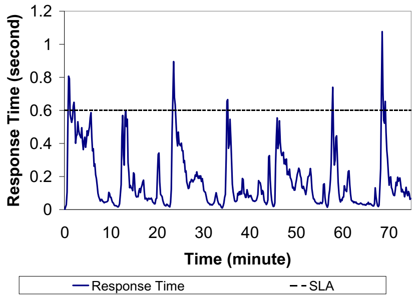
Figure 3. Comparison of the Proactive and Reactive provisioning schemes



(a) Load Function



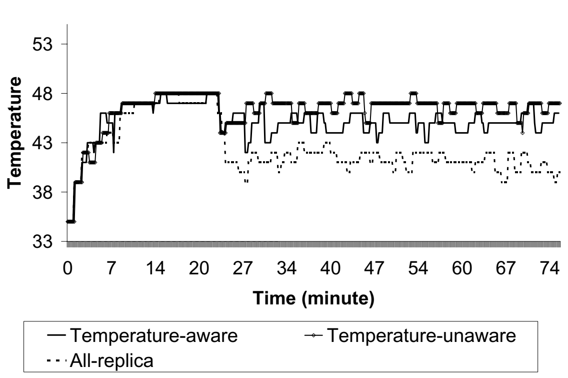
(b) Replica Allocation



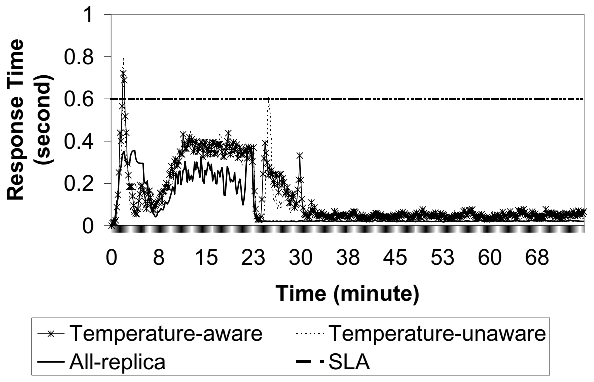
(c) Average Latency

Figure 4. Adapting to Changes in Workload Mix

in Figure 6(b), the latency for all the three schemes is higher than before. The temperature profile depicted in Figure 6(a) shows that the all-replica scheme has the worst (highest) temperature while the temperature-aware scheduling and the baseline temperature-unaware scheme result in the best and second-best temperature profiles respectively. Unlike in the previous scenario, the CPU utilization is high and increases the CPU temperature for all machines. The high temperature in the all-replica load spread scheme is due to allocating significant load to machines on the top

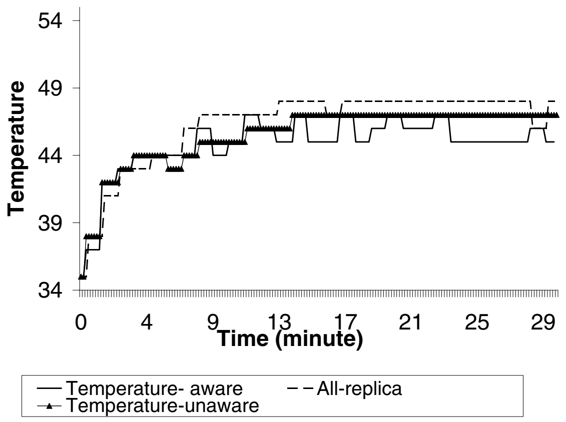


(a) Temperature

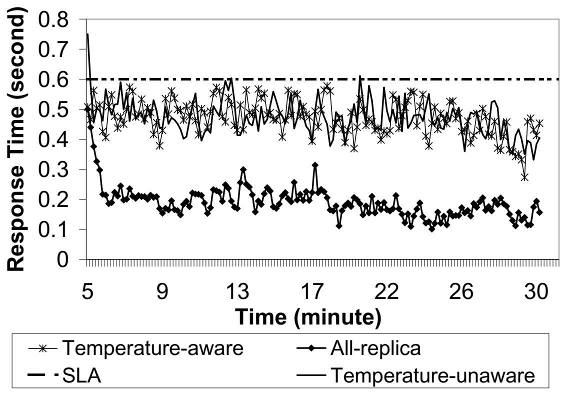


(b) Average Latency

Figure 5. Temperature Profile for Lightly Loaded Cluster



(a) Temperature



(b) Average Latency

Figure 6. Temperature Profile for Constrained Number of Replicas

of the cluster rack, which receive warmer inlet air, hence experience sharper temperature increases with load. Indiscriminately spreading load on all machines results in hot spots on these machines. This experiment shows that for constrained resource cases, temperature-aware load-shifting achieves better temperature profile while maintaining SLA requirements.

8 Related Work

In this paper, we investigate autonomic resource provisioning based on on-line learning, pursuing three orthogonal objectives: meeting a performance SLA, optimizing resource allocation, and improving the temperature profile in a cluster of database servers. Related work in the area of database provisioning based on learning techniques includes our previous KNN-based provisioning of database replicas[7]. This previous scheme requires extensive off-line training [7] of system states for each representative workload mix in order to build an accurate performance model. A similar table-driven provisioning approach [25] stores response time values from off-line experiments with different workload density and different numbers of servers. A simple interpolation is used to obtain missing values in the table. Just like our previous KNN-based technique, this approach needs a large amount of data in order to be able to interpolate accurately when the workload mix changes.

Other schemes based on analytical models [5, 26, 18] have also demonstrated good accuracy in simulation, or experimentally for provisioning state-less servers e.g., web servers. The general problem associated with these approaches is that the analytical model may be expensive to build for complex systems and workloads such as those of database servers and can become inaccurate for unforeseen changes in environment or workload mix.

Cohen et al. [8] and Zhang et al. [28] use a similar approach for estimating performance models by means of statistical learning. They use tree-augmented Bayesian networks (TAN) to discover correlations between system metrics and service level objectives (SLO). Through training, the TAN discovers the subset of system metrics that lead to SLO violations. While this approach predicts violations and compliances with good accuracy, it does not provide any information on how to adapt to avoid SLO violations. It also requires extensive off-line training. In contrast, our provisioning scheme determines the appropriate database configuration based on a dynamic performance model.

Our temperature-aware enhancements for the provisioning scheme draw on previous work describing simulations based on thermodynamic concepts [27]. They show that IT level provisioning can significantly improve the way heat is generated and delivered to cluster room air conditioning. In our case, replica consistency considerations prevent us from using arbitrary machines for achieving generic temperature related objectives. In the same way, related work [6, 19] in the area of heat management by shifting load differs from ours in that the proposed solutions are not specific to database clusters. Furthermore, our work presents our experience with a prototype implementation deployed on a live cluster environment instead of simulations.

9 Conclusions

In this paper, we propose a novel autonomic provisioning scheme for the database back-end of dynamic content web servers. The novel aspects of our proactive provisioning scheme are: i) building a lightweight performance model on-line by tracking the correlation between selected system metrics, the database configuration and the resulting query latency, ii) allocating database replicas in advance of need based on the dynamic performance model and load predictions, and iii) shifting the load from hot-spot nodes to newly allocated low-temperature nodes.

Our provisioning scheme uses on-line learning and adapts its model to workload mix and environment changes. Compared to instance-based learning schemes such as K-Nearest-Neighbors or tree-based classifiers, which need to maintain a large set of samples, our method based on Support Vector Machines exhibits high extrapolation capabilities based on a small sample set.

We show that a temperature-aware provisioning scheme with load shifting maintains the query latency under the service level agreement, while having better resource usage compared to a uniform load spread scheme which uses all machines available. At the same time, this provisioning scheme reduces the temperature of the node with the highest temperature, thus affording cooling cost savings compared to a provisioning scheme that is not temperature aware.

References

[1] The Apache Software Foundation. http://www.apache.org/.

[2] C. Amza, A. L. Cox, and W. Zwaenepoel. Distributed versioning: Consistent replication for scaling back-end databases of dynamic content web sites. In M. Endler and D. C. Schmidt, editors, *Middleware*, volume 2672 of *Lecture Notes in Computer Science*, pages 282–304. Springer, 2003.

[3] M. F. Arlitt and C. L. Williamson. Internet web servers: workload characterization and performance implications. *IEEE/ACM Trans. Netw.*, 5(5):631–645, 1997.

[4] Y. Baryshnikov, E. Coffman, G. Pierre, D. Rubenstein, M. Squillante, and T. Yimwadsana. Predictability of web-server traffic congestion. In *WCW '05: Proceedings of the 10th International Workshop on Web Content Caching and Distribution (WCW'05)*, pages 97–103, Washington, DC, USA, 2005. IEEE Computer Society.

[5] M. N. Bennani and D. A. Menascé. Resource allocation for autonomic data centers using analytic performance models. In *ICAC*, pages 229–240. IEEE Computer Society, 2005.

[6] D. J. Bradley, R. E. Harper, and S. W. Hunter. Workload-based power management for parallel computer systems. *IBM J. Res. Dev.*, 47(5-6):703–718, 2003.

[7] J. Chen, G. Soundararajan, and C. Amza. Autonomic provisioning of backend databases in dynamic content web servers. In *Proceedings of the Third International Conference on Autonomic Computing (ICAC 2006)*, pages 123–133, 2006.

[8] I. Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *OSDI*, pages 231–244, 2004.

[9] K. Coleman, J. Norris, G. Candea, and A. Fox. Oncall: Defeating spikes with a free-market server cluster. In *In Proceedings of the 1st International Conference on Autonomic Computing (ICAC)*, 2004.

[10] H. Drucker, C. J. C. Burges, L. Kaufman, A. J. Smola, and V. Vapnik. Support vector regression machines. In M. Mozer, M. I. Jordan, and T. Petsche, editors, *NIPS*, pages 155–161. MIT Press, 1996.

[11] T. Hastie, R. Tibshirani, and J. H. Friedman. *The Elements of Statistical Learning*. Springer, August 2001.

[12] IBM Corporation. Automated provisioning of resources for data center environments. http://www-306.ibm.com/software/tivoli/solutions/provisioning/, 2003.

[13] A. K. Iyengar, M. S. Squillante, and L. Zhang. Analysis and characterization of large-scale web server access patterns and performance. *World Wide Web*, 2(1-2):85–100, 1999.

[14] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. N. Tantawi. Dynamic placement for clustered web applications. In L. Carr, D. D. Roure, A. Iyengar, C. A. Goble, and M. Dahlin, editors, *WWW*, pages 595–604. ACM, 2006.

[15] D. A. Menascé, D. Barbará, and R. Dodge. Preserving qos of e-commerce sites through self-tuning: a performance model approach. In *ACM Conference on Electronic Commerce*, pages 224–234. ACM, 2001.

[16] J. Moore, J. Chase, P. Ranganathan, and R. Sharma. Making scheduling "cool": Temperature-aware resource assignment in data centers. In *USENIX Annual Technical Conference*, pages 61–75, 2005.

[17] MySQL. http://www.mysql.com.

[18] D. Narayanan, E. Thereska, and A. Ailamaki. Continuous resource monitoring for self-predicting dbms. In *MASCOTS '05: Proceedings of the 13th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 239–248, Washington, DC, USA, 2005. IEEE Computer Society.

[19] S. Osman, D. Subhraveti, G. Su, and J. Nieh. The design and implementation of zap: a system for migrating computing environments. *SIGOPS Oper. Syst. Rev.*, 36(SI):361–376, 2002.

[20] PHP Hypertext Preprocessor. http://www.php.net.

[21] S. M. Ross. *Introduction to Probability and Statistics for Engineers and Scientists*. John Wiley & Sons, Inc., 1987.

[22] G. Soundararajan, C. Amza, and A. Goel. Database replication policies for dynamic content applications. In *In Proceedings of the First ACM SIGOPS EuroSys*, 2006.

[23] G. Tesauro, R. Das, W. E. Walsh, and J. O. Kephart. Utility-function-driven resource allocation in autonomic systems . In *ICAC*, pages 70–77, 2005.

[24] Transaction Processing Council. http://www.tpc.org/.

[25] W. E. Walsh, G. Tesauro, J. O. Kephart, and R. Das. Utility functions in autonomic systems. In *In Proceedings of the 1st International Conference on Autonomic Computing (ICAC)*, 2004.

[26] M. Woodside, T. Zheng, and M. Litoiu. Service system resource management based on a tracked layered performance model. In *Proceedings of the Third International Conference on Autonomic Computing (ICAC 2006)*, pages 123–133, 2006.

[27] P. R. YJ. Moore, J. Chase and R. Sharma. Making scheduling cool: Temperature-aware resource assignment in data centers. In *Proceedings of USENIX*, 2005.

[28] S. Zhang, I. Cohen, M. Goldszmidt, J. Symons, and A. Fox. Ensembles of models for automated diagnosis of system performance problems. In *DSN*, pages 644–653. IEEE Computer Society, 2005.