Cost-Aware Dynamic Provisioning for Performance and Power Management

by

Saeed Ghanbari

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

Abstract

Cost-Aware Dynamic Provisioning for Performance and Power Management

Saeed Ghanbari

Master of Science

Graduate Department of Electrical and Computer Engineering

University of Toronto

2008

Dynamic provisioning of server boxes to applications entails an inherent performance-power trade-off for the service provider, a trade-off that has not been studied in detail. The optimal number of replicas to be dynamically provisioned to an application is ultimately the configuration that results in the highest revenue. The service provider should thus dynamically provision resources for an application only as long as the resulting reward from hosting more clients exceeds its operational costs for power and cooling. We introduce a novel cost-aware dynamic provisioning approach for the database tier of a dynamic content site. Our approach employs Support Vector Machine regression for learning a dynamically adaptive system model. We leverage this lightweight on-line learning approach for two cost-aware dynamic provisioning techniques. The first is a temperature-aware scheme which avoids temperature hot-spots within the set of provisioned machines, and hence reduces cooling costs. The second is a more general cost-aware provisioning technique using a utility function expressing monetary costs for both performance and power.

Contents

Chapter 1

Introduction

The operational costs involved in managing large-scale dynamic content servers currently exceed 77% of the average web service provider budget [39]. Dynamic provisioning of resources to all tiers in a dynamic content web site [8, 18, 19, 37] has been proposed to alleviate this problem. Previous work on dynamic provisioning focuses only on performance aspects. Specifically, previous pro-active provisioning solutions maintain spare idle machines or even pre-warm spare machines for the application workload to minimize the adaptation delay [35]. While the performance penalty to applications is minimized, the power and cooling costs of the additional replicas could be high. Thus, as promising as it may be for server self-management, dynamic provisioning brings with it an inherent trade-off between optimizing performance on one hand, and reducing power/cooling costs on the other hand, trade-off that has not been previously studied.

In this work, we investigate new cost-aware pro-active dynamic provisioning techniques in the database back-end tier of a dynamic content web site. Transparent automatic provisioning in this tier is difficult due to the challenge of accurately modeling the behavior of a replicated database cluster. An additional challenge is the inherently higher adaptation delay of adding a database replica, compared to the latency of adding a server in a state-less tier, such as the Web server tier.

We introduce and experimentally evaluate a novel unifying approach to performance and power management of a replicated database tier. For this purpose, we use online adaptive tracking (OAT) of metric correlations. The key novel feature of OAT is on-line learning of correlations between system states and higher level application goals, such as, 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 low-level system metrics and configurations and their corresponding application-level metrics. The system accumulates a set of sample points considered representative for the learned correlation function and adjusts this set of sample points on-the-fly when necessary.

The learned correlations in OAT form the basis for building a dynamic model of the system that adapts on-line and provides guidance for adding and removing database replicas in the database tier to meet higher level goals. In this work, we use and evaluate OAT in conjunction with two higher level goals: i) combining performance and temperature awareness for providing latency according to a Service Level Agreement (SLA), while reducing cooling costs and ii) combining performance and power awareness for optimizing overall monetary costs. To achieve the former goal, we enhance OAT-driven dynamic provisioning with a scheduling technique that avoids temperature hot-spots within already provisioned replicas [16]. For the latter goal, we use OAT with a utility function that maps directly to monetary costs. We observe that the optimal dynamic provisioning approach depends on the relative performance/power costs for the service provider. Intuitively, at one extreme, if the monetary reward for running a particular application does not exceed the power and cooling costs, the approach should not provision any replicas to the respective application. At the other extreme, if power costs are negligible compared to the revenue obtained for running a particular application, then the ideal approach may over-provision resources for the respective application. This more general cost-aware provisioning technique uses OAT to build a dual performance/power model based on a

specified monetary utility. It uses this model to drive adaptations to workload spikes. Physical servers are in power-saving stand-by mode if there is no work allocated to them. A physical server is brought on-line and updated proactively for a given application based on the learned cost/benefit outcome of provisioning.

In our prototype implementation, we use *Support Vector Machine* (SVM) [15] regression for determining the correlation function. 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 or dynamic changes in the application and system, such as a workload mix shift, a change in database hardware, etc. We use minimal off-line training to guide the selection of the most promising correlations that the system should automatically determine on-line.

We evaluate our pro-active schemes with and without enhancements using the shopping and browsing mix workloads of the TPC-W e-commerce benchmark, 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. We compare the provisioning technique with on-line learning against a baseline reactive provisioning technique [36]. We also compare our temperature-aware and monetary cost-aware provisioning techniques against baseline techniques without cost-awareness.

Our results are as follows:

1. Our correlation tracking is shown to be accurate, translating into advance actuation of resources and improved performance for the proactive scheme compared to the baseline reactive scheme. Correlation tracking is also shown to be robust to on-line workload mix changes.

2. Our temperature-aware load shifting scheme is able to keep the maximum temperature in the cluster lower than temperature-unaware schemes, thus avoiding hot-spots. At the same time the load-shifting temperature-aware scheme has similar performance to the temperature-unaware schemes, while using fewer resources than the uniform load spread technique.

3. The utility-based proactive technique adapts towards database configurations that maximize the monetary profit of the service provider for different power/performance cost scenarios in the data center.

The rest of the dissertation is organized as follows. We first introduce the necessary background related to dynamic provisioning and the SVM learning algorithm in Chapter 2. We then introduce our system architecture and give a brief overview of our dynamic replication environment in Chapter 3. Then, we discuss our pro-active approach based on on-line learning in Chapter 4 and the two cost-aware enhancements, temperature-awareness and monetary cost-awareness, in Chapter 5, and Chapter 6, respectively. We describe a baseline reactive provisioning approach used for comparison in the experiments in Chapter 7. Chapter 8 illustrates our results for applying our approaches in the adaptation process of database clusters under different workload patterns and dynamically varying workload mixes. We compare our work to related work in Chapter 9. Finally, we conclude in Chapter 10.

Chapter 2

Background

In the following, we introduce background in dynamic provisioning for dynamic content servers and Support Vector Machines, which our adaptive learning provisioning solution builds on.

2.1 Dynamic Provisioning in Dynamic Content Servers

Dynamic content servers, such as, Amazon.com and eBay.com, commonly use a three-tier architecture (see Figure 2.1) 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 the dynamic content of the site.

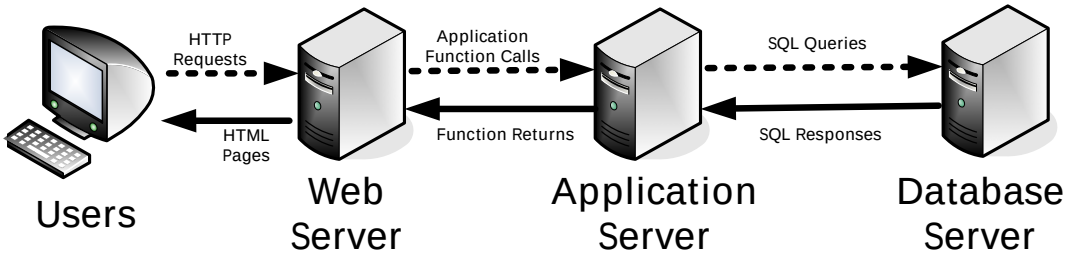


Figure 2.1: Three-tier Architecture

Large data centers may host multiple applications concurrently, such as, e-commerce, auc-

tions, news and games. The cooling and power costs of gross hardware over-provisioning for each application's estimated peak load are making efficient resource usage crucial. Furthermore, the excessive personnel costs involved in server management motivate an automated approach to resource allocation for applications in large sites.

Dynamic resource allocation techniques, i.e., dynamic provisioning of servers to multiple application in each tier of the dynamic content site, have been recently introduced to address the increasing costs of ownership for large dynamic content server clusters. These automatic solutions add servers to an application's allocation based on perceived or predicted performance bottlenecks caused by either load spikes or component failures; they remove resources from a application's allocation when in underload.

If services experience daily patterns with peak loads for each service type at a different time (e.g., daytime for e-commerce, evening for auctions, morning for news sites, night for gaming), there are opportunities for re-assigning hardware resources from one service to another. Thus, instead of gross hardware over-provisioning for each application's estimated peak load, dynamic resource provisioning techniques enable the service provider to efficiently multiplex data center resources across applications.

2.1.1 Common Architecture for Database Replica Provisioning

Figure 2.2 shows the common architecture of sites with dynamic provisioning in the database server tier. A resource manager makes the replica allocation decisions for each application hosted on the site based on the application requirements and the current system state. The requirements are expressed in terms of a service level agreement (SLA) that consists of a latency requirement on the application's queries. The current system state includes the current performance of this application and the system capacity. The resource manager operates in two modes, underload and overload. During underload, the number of replicas exceeds overall demand and allocation decisions per application are made independently. During overload, the manager uses either a utility-based scheme e.g., profit-based, or a fairness scheme e.g., equal

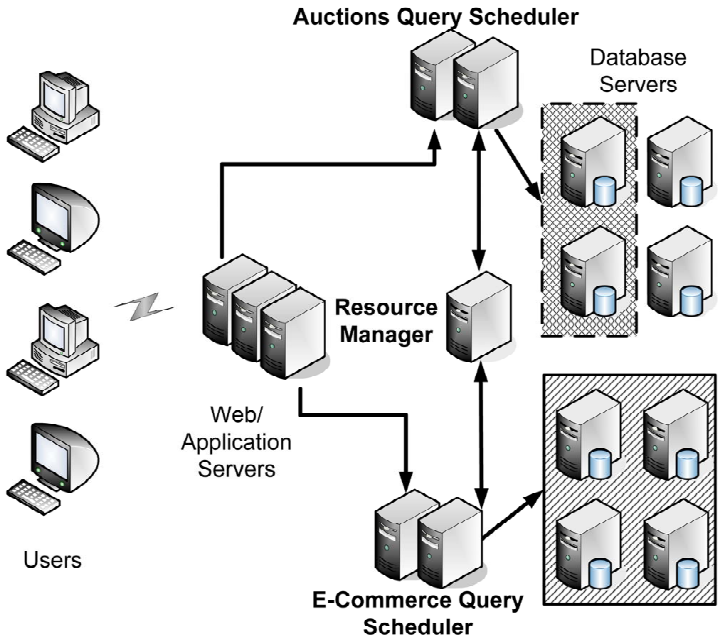


Figure 2.2: Cluster Architecture

share, to allocate replicas to applications.

The allocation decisions are communicated to a set of schedulers, one per application, interposed between the application and the database tiers. Each scheduler fulfills the following functions: i) It keeps track of the current *database set* allocated to its application and allocates or removes replicas from its managed set according to the resource manager's decisions, ii) It distributes the corresponding incoming requests onto the respective database replicas, iii) It samples various system and application metrics, e.g., the average application latency, periodically from its *database set* in order to perceive or predict resource bottlenecks, and iv) It provides consistent replication, i.e., one-copy serializability [9], at all the replicas allocated to the application it manages.

Strong consistency is desirable for dynamic provisioning in the database tier of a multi-tier data center, for transparency reasons; the replicated nature of the database back-end and its configuration adaptations are thereby hidden from the application server, which interacts with the replicated back-end as with a single database. Any eager replication scheme with one-copy

serializability [9] can be used within each application's replica set. However, existing dynamic provisioning schemes typically leverage the presence of the scheduler and the synchronous, one query at a time, nature of the communication between the application and database tiers in a data center to implement a middleware replication solution in the scheduler itself. Upon receiving a query from the application server, the scheduler sends the query using a read-one, write-all replication scheme to the replica set allocated to the application. The scheduler assigns a global serialization order to all transactions pertaining to its application based on conservatively-perceived table-level conflicts between transactions, and ensures that transactions execute in this order at all the corresponding database replicas. Table-level concurrency control affords optimizations based on conflict awareness in the scheduler [4, 2], which offset any penalties due to the coarse-grain control for read-intensive e-commerce applications [35].

The scheduler is also in charge of bringing a new replica up to date by a process called *data migration*, during which all missing updates are applied on that replica. Integrating a stale replica into an application's allocation occurs through an on-line reconfiguration technique [42, 35] without stalling on-going transactions on already active replicas for that application. Finally, each scheduler may itself be replicated for availability [4, 2].

2.1.2 Scaling

Amza et al. [4, 2, 3] have shown that database replication on commodity clusters scales well and is a viable alternative to using expensive multiprocessor and/or network storage hardware configurations. Next, we present some results from this work to motivate our dynamic replication approach. Figures 2.3 and 2.4 show the performance scaling graphs for the TPC-W and the RUBIS benchmarks on a cluster of 8 database machines.[1] These graphs show that throughput scales almost linearly for 8 machines.

To explore scaling with larger clusters, Figures 2.5 and 2.6 show the simulated performance scaling graphs for TPC-W and RUBIS with 60 replicas. These numbers are within 12% of the

[1]The machines are AMD Athlon MP 2600+ computers with 512MB of RAM, using RedHat Fedora Linux.

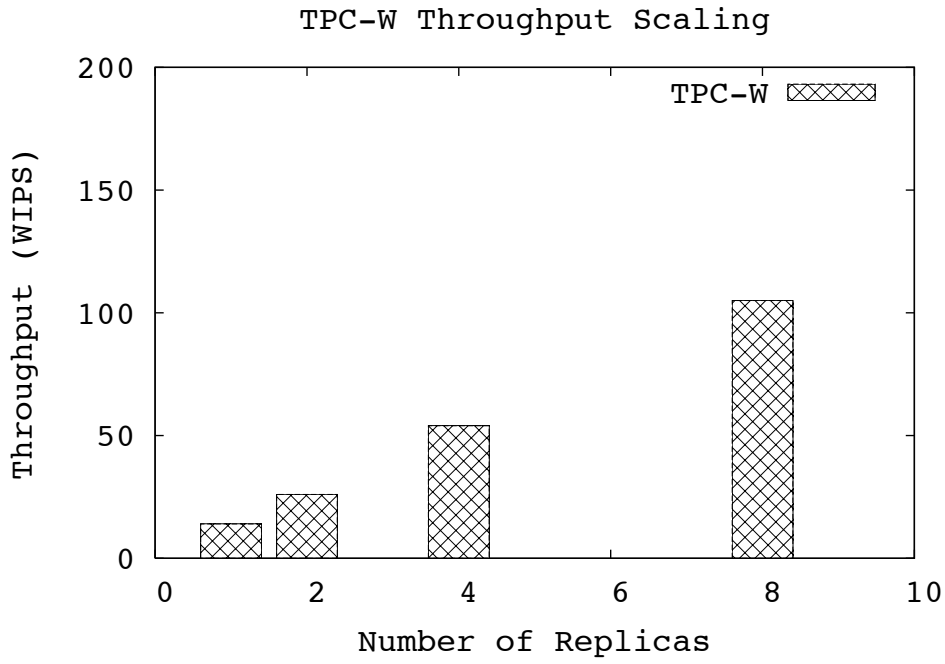


Figure 2.3: TPC-W throughput with increasing replication

experimental numbers for both applications. These simulations show that TCP-W scales better than RUBIS. The reason is that replication allows scaling the throughput of read queries but can slow down write queries, and the time spent in executing reads versus writes is 50 to 1 for TPC-W and 8 to 1 for Rubis. These graphs show that the knee of the scaling curve depends on the workload and cannot be determined apriori. With our dynamic replication approach, each application could be assigned replicas based on the scaling curve. These result also indicates that relation between number of replicas and performance in non-linear.

2.1.3 Overview of Dynamic Provisioning Solutions

Several fully-transparent provisioning solutions [8, 19, 37] have been recently introduced to address the increasing cost of management problem. Many of these approaches [32, 23, 19, 8,

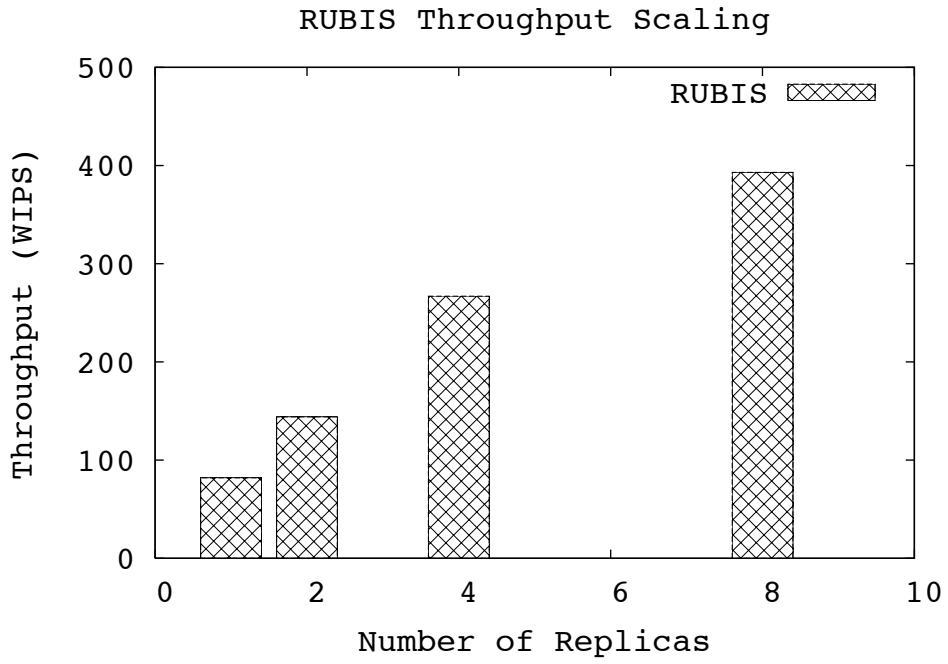


Figure 2.4: RUBIS throughput with increasing replication

41, 24, 40, 37] investigate dynamic provisioning of resources within the (mostly) stateless web server and application server tiers. In this article, we focus on the dynamic resource allocation solutions within the stateful database tier, which commonly becomes the bottleneck [1, 34].

Regardless of the tier it applies to, a dynamic provisioning solution can be either *proactive* or *reactive* depending on its capabilities for predicting performance bottlenecks in the dynamic content server. *Proactive* solutions use sophisticated system models for prediction, such as, queuing models [8, 41], utility models [37, 40], machine learning models [16, 12] or market-place approaches [14]. *Proactive* provisioning techniques use the system model predictions for triggering allocations in advance of expected need. This is especially important for tiers with a higher adaptation delay, such as the database tier. In contrast, *reactive* approaches do not use prediction, but rather detect and react to existing resource bottlenecks. They rely on predefined thresholds for application-level or system metrics, such as latency or CPU usage for triggering

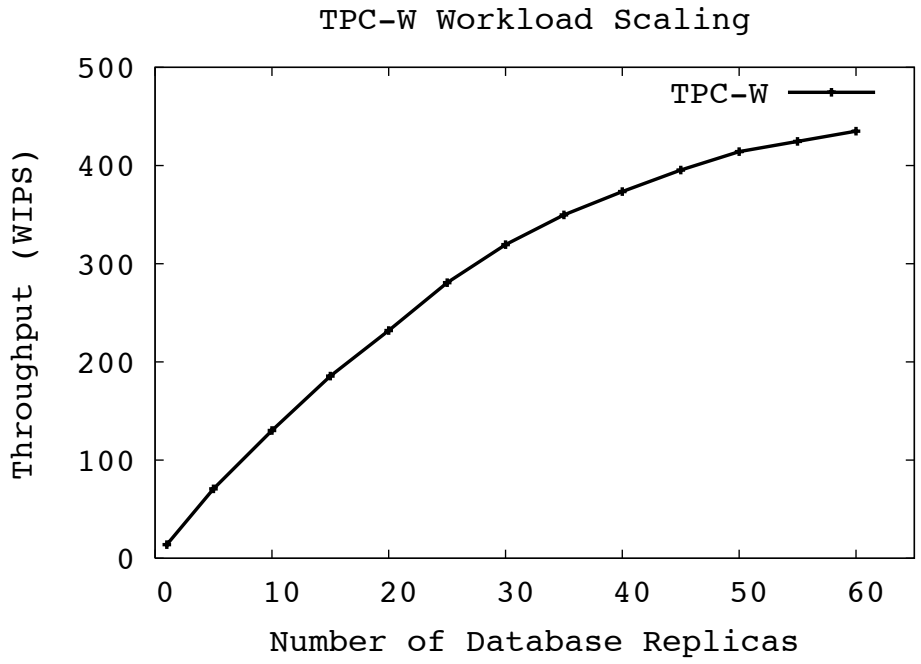


Figure 2.5: TPC-W throughput (simulation)

changes in application allocation.

Proactive approaches typically do not model multiple database replicas as part of their solution search space [8, 22, 25, 40, 41].

2.2 Challenges for Database Replica Provisioning

2.2.1 Adaptation Delay

Adapting to a bottleneck caused by either a workload spike or a failure by allocating additional replicas to the application is not an immediate process. The database state of the new replica(s) for that application will be stale and must be brought up-to-date, or a new instance of that application may need to be installed before it can be used. Furthermore, load balacing for the

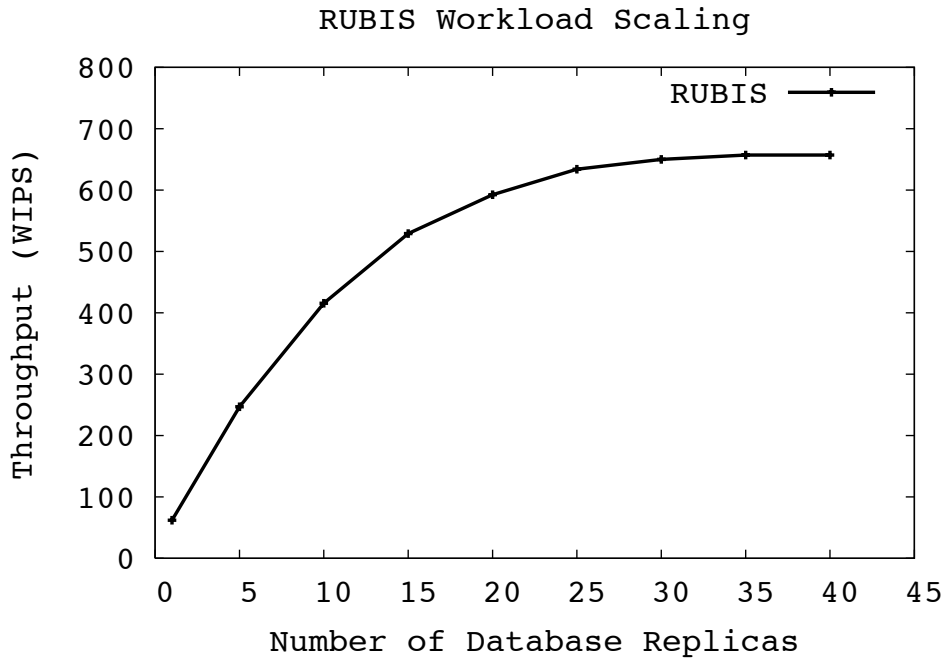


Figure 2.6: RUBIS throughput (simulation)

old and new replicas needs to occur and the buffer pool at the new replica(s) needs to be warm before the new replica(s) can be used effectively.

Potential for Oscillations in Allocation

Oscillations in database allocations to applications may occur during system instability induced by adaptations. As discussed earlier, the replica addition process can be long. *During* the adaptation phases, i.e., data migration, buffer pool warmup and load stabilization, the latency will remain high or may even temporarily continue to increase as shown in Figure 2.7. Latency sampling during this potentially long time is thus not necessarily reflective of a continued increase in load, but of system instability after an adaptation is triggered. If the system takes further decisions based on sampling latency during the stabilization time, it may continue to add further replicas which are unnecessary, hence will need to be removed later. This is an

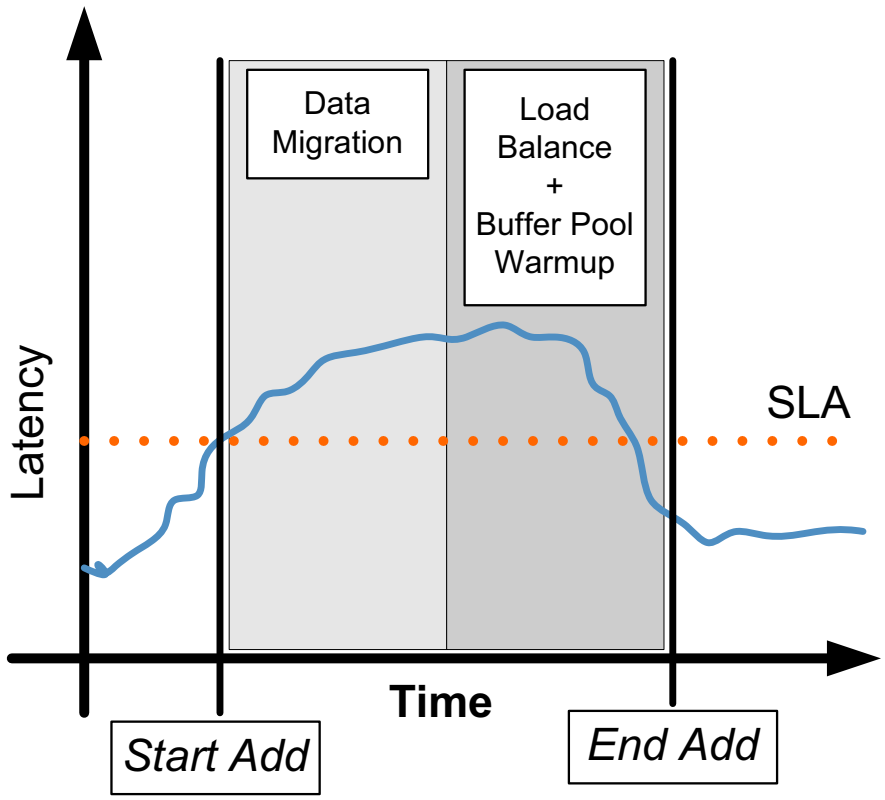


Figure 2.7: Latency instability during replica addition.

oscillation in allocation which carries performance penalties for other applications running on the system due to potential interference.

Either a reactive or proactive policy that measures application-level metrics, periodically, including during the replica addition process, can suffer from allocation instability. Allocation oscillations, in their turn, cause cross-application interference due to the price paid for warming up the buffer pool as part of the "context-switch" between applications on the machines involved.

While rapid load fluctuations may induce similar behavior, simple smoothing or filtering techniques can offer some protection to very brief load spikes. All existing dynamic provisioning schemes use some form of smoothing or filtering, to dampen brief load fluctuations.

2.2.2 Accurate and Lightweight Modelling

As previously mentioned proactive provisioning is desirable in the database back-end tier due to the higher adaptation delay inherent in this tier. The challenge for proactive database tier provisioning lies in accurately modeling the behavior of a replicated database back-end tier. Many factors, such as the wide range of query execution times typical in e-commerce workloads, the load balancing policy, caching effects, the replica consistency maintenance algorithm, etc. may influence performance. The derivation of a classical analytical model taking into account even a subset of these factors can be time consuming, hence unsuitable for on-line modeling and adaptation.

Data Migration. Data migration [36] is an on-line technique for bringing a new replica up to date with minimal disruption of transaction processing on existing replicas in the application's allocation.

Each scheduler maintains persistent logs for all write queries of past transactions in its serviced workload for the purposes of enabling dynamic data replication. The scheduler logs the queries corresponding to each update transaction and their version numbers at transaction commit. The write logs are maintained per table in order of the version numbers for the corresponding write queries.

During data migration for bringing a stale database replica up-to-date, the scheduler replays on it all missing updates from its on-disk update logs. The challenge for implementing an effective data migration is that new transactions continue to update the databases in the workload's allocation while data migration is taking place. Hence, the scheduler needs to add the new database replica to its workload's replica mapping *before* the end of data migration. Otherwise, the new replica would never catch up. Unfortunately, new updates cannot be directly applied to the (stale) replica before migration completes. Hence, new update queries are kept in the new replica's holding queues during migration. In order to control the size of these holding queues, the scheduler executes data migration in *stages*. In each stage, the scheduler

reads a batch of old updates from its disk logs and transfers them to the new replica for replay without sending any new queries. Except for pathological cases, such as sudden write-intensive load spikes, this approach reduces the number of disk log updates to be sent after each stage, until the remaining log to be replayed falls below a threshold bound. During this last stage, the scheduler starts to send new updates to the replica being added, in parallel with the last batch of update queries from disk.

2.3 Interaction between Resource Provisioning and Power Consumption

Server machines and cooling facilities are major power consumers in large data centers. The cooling cost is a function of both server heat production and the air-flow physics in the server room. Power consumption by a server can be controlled at various granularities, ranging from CPU throttling to turning off the machine. Typically, turning off a machine or parts of it, i.e., either shutting down the whole server or only the CPU and peripherals, results in superior power conservation compared to finer grain power control mechanisms. A shut-down server produces no heat, hence no cooling costs. Placing whole servers in *stand-by mode* results in similar power and cooling savings. For example, we performed some preliminary experiments on a cluster of dual CPU PE 1830 Xeon servers. A server blade from this cluster consumes 177 Watts at a corresponding CPU temperature of around 35°C, when turned on, but idle, while it consumes only 4.24 Watts at a CPU temperature equal to the cluster room temperature of 20°C, when it is in standby mode.

Dynamic provisioning can substantially reduce cost due to power savings ; to this end, machines that are not in use should be powered down or put in stand-by mode for maximum power savings. On the other hand, the need for the power-up cycle for bringing a machine on-line, increases the adaptation delay, resulting in potential *cost increases*; a workload under a sudden load spike may violates its Service Level Agreement (SLA) hence monetary penalties

to the service provider. This motivates the need for a pro-active provisioning scheme using a learning approach to system modeling as we investigate in our work. In the following, we introduce background on the SVM machine learning algorithm we build on.

2.4 Support Vector Machines (SVM)

We use Support Vector Machines (SVM) [15] 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:

$$f(X) = \sum_{i=1}^{l} \alpha_i y_i K(X_i,X) \tag{2.1}$$

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.

Chapter 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.

3.1 Adaptation Process

Since replicas are stateful entities, allocating a replica to an application can introduce a high adaptation delay. A database replica can be in one of the following five states:

1. *Stand-by mode* : The server disk, memory and processor are turned off. At shut-down, an image of the memory is written to disk, for the purposes of restoring the server state when turned back on.

2. *Stale*: The server has been turned on in preparation for allocation, but does not belong to the application allocation yet, hence receives no updates for the respective application.

3. *Periodically Updated*: The scheduler sends batch updates to the newly added replica, periodically, in preparation for adding the replica to the write-set.

4. *Write-set*: The replica receives all update queries for the application.

5. *Read-set*: The replica receives, and executes all update queries as it participates in load balancing of read query execution for the application.

Corresponding to a replica state, the replica addition process consists of four phases: i) bringing the server on-line if it is currently in stand-by mode, ii) data migration to bring the server up to date, iii) adding the replica to the application's write-set and iv) adding the replica to the application's read set and waiting for system stabilization.

Bringing a server on-line implies turning the server on and restoring its state from the image stored on disk. Data migration involves applying logs of missing updates on the new replica to bring it up-to-date. System stabilization involves load balancing and warmup of the buffer pool on the new replica. A problem brought about by the period of instability that occurs during adaptations is that 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 [12]. Hence, all our provisioning techniques explicitly recognize the system stabilization phase as part of the adaptation process.

While some of the stages above may overlap, replica addition as a whole can introduce a long period over which query latencies are high. For example, just the first phase (turning on a server and restoring its image) takes around 120 seconds on the server cluster we use in our experiments. Hence, our provisioning techniques use a set of intermediate states between the turned-off and fully integrated state, such as the *periodically updated* intermediate state, to have more control in adaptation actuation and prepare servers for addition in advance of predicted need.

Chapter 4

Proactive Provisioning with On-Line Learning

In this chapter, we introduce the elements of our provisioning scheme with on-line learning which is used in our temperature-aware, and utility-based proactive provisioning approaches.

Our provisioning scheme has three fundamental elements: a Load predictor, an Online Adaptive Tracking model, and a Provisioning module. The idea is to build a model of the system which relates workload metrics to the metrics of interest used to estimate the high level application goals. In this chapter, we introduce a model that uses only performance as the high level objective. In later chapters, we show how the same technique can be expanded to include cost-based goals, such as temperature optimization or a generic application utility as well. Irrespective of the higher level goal, the high level metric can be predicted based on the model and the load prediction. Finally, the provisioning module uses the model and the workload prediction to actuate addition and removal of replicas in such a way that to optimize the high level objectives.

Our workload predictors are based on linear extrapolation. Our performance model generation is based on a novel correlation detection scheme, Online Adaptive Tracking (OAT), for correlating load intensity with the higher level metrics of interest.

In the following sections we describe each of the three elements of our scheme assuming a performance goal, i.e., maintaining query latency under a given SLA, in more detail.

4.1 Dynamic Model Generation

The key technique in our dynamic provisioning approach is Online Adaptive Tracking (OAT). OAT correlates system metrics to the application-level metrics of interest based on dynamically generated performance models of the system. Specifically, our OAT approach uses Support Vector Machine (SVM) regression to generate dynamic performance models 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 metric of interest.

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.

Our Online Adaptive Tracking (OAT) algorithm uses a set of samples of observed workload metrics, their corresponding database configurations and the target metric for obtaining the correlation function.

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 thresh-

old. 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 K closest samples in the sample set, and replaces them with a linear combination of the old and new samples by the following formula:

$$(y^k_{new}, \vec{x}^k_{new}) \leftarrow \mu(y^k_{old}, \vec{x}^k_{old}) + (1-\mu)(y_{actual}, \vec{x}_{cur}) \forall k \in K \tag{4.1}$$

Closeness is defined as $l_p - norm$ of difference of x_{cur} and x_{old} part of samples. 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 4.1 evolves the sample set so that recent changes in the relation of $\vec{x}$ and y are reflected in the sample set. On the other hand, this evolution is done gradually, such that an errant measurement cannot significantly influence the sample set.

4.2 Load Predictor

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 [20, 6, 7] 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 [17] 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.

4.3 Performace-Based 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 provisioning 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 Provisioning module is then responsible for instructing the scheduler of each application when and how to add or remove databases.

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 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 sampling interval. Instability is detected when the throughput variation is inversely correlated with the response time variation, e.g., a decrease in throughput coincides with an increase in response time.

4.3.1 Feature Selection for Tracking Correlations

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 [33] and cross-validation [17]. 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.

Chapter 5

Temperature-Aware Proactive Provisioning

In this chapter, we introduce an enhancement to our performance-driven pro-active provisioning scheme based on temperature-aware scheduling. This scheme has a dual performance-temperature optimization objective. The primary goal is to maintain the query latency below the SLA as before. As a secondary goal, we add temperature-aware scheduling for minimizing the temperature of the hottest machine within the set of machines allocated to each application. Previous work [26] 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 introduced in the previous chapter 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.

Chapter 6

Utility-Based Proactive Provisioning

We extend our proactive provisioning scheme to consider a generic application utility instead of performance as the high level goal. The application utility is a user defined function which indicates the benefit gained by allocating replicas to an application, as well as the cost it incurs. Our scheme provides a framework to proactively allocate replicas in such a way that the utility function is maximized under a changing workload.

As before, the provisioning scheme uses a workload predictor, a dynamic system model derived through on-line adaptive tracking of correlations and a provisioning module. In this case, our approach learns correlations between system configurations/metrics and the associated utility. It uses the learned correlations to predict the best configuration to use and actuates addition and removal in advance of predicted need.

The utility function we use translates both performance and power objectives into monetary costs. Intuitively, monetary costs is the objective of ultimate importance to the service provider. In this way, our proposed scheme provides a unified framework to actuate addition and removal of replicas according to a single overall cost-based objective. Our dynamic provisioning technique is, however, independent of the particular utility function we use.

In the following we present the utility function we use, which is not the only feasible definition, and then the design of our provisioning module.

6.1 Utility Function

Our utility function consists of two components called *power-utility* and *performance-utility*, corresponding to the two factors that impact the overall utility.

We estimate the power consumption by a linear relation with the average CPU utilization, given the minimum and maximum power consumption corresponding to the idle state, and the fully utilized server, respectively. Our load-balancing scheme ensures that read queries are uniformly distributed among servers in the read-set, resulting in similar average CPU utilization among the servers allocated to an application. Accordingly, we evaluate *power-utility* as follows:

$$U^{pow}(nc, n_r, n_w) = -C_{powLoss} \times n_r \times (P_{min} + CPU(nc, n_r) \times (P_{max} - P_{min}) + P_{min}(n_s - n_r)) \tag{6.1}$$

where nc is the predicted number of clients, according to a load predictor, $C_{powLoss}$ is the power cost, P_{min} and P_{max} are the minimum and maximum server power consumption according to the server specification, $CPU(nc, n_r)$ is an OAT function estimating the server's CPU utilization given the number of clients and the number of database replicas in the read-set, and finally n_s, and n_r are the number of machines allocated to the application, and the number of machines in the read-set, respectively. In the formula, the power consumption of the idle machines is P_{min} and their number is computed as the difference between n_s and n_r.

For defining the impact of performance on the utility function, i.e., the *performance-utility*, we need to consider two factors: the benefit gained by delivering services compliant to the SLA, and the penalties imposed by failing to meet the SLA. Accordingly, for the *performance utility* we use the following formulas:

$$R(nc, n_r) = \frac{nc}{tx(nc, n_r)} - Z \tag{6.2}$$

$$U^{perfGain}(nc, n_r, n_w) = [SLA - R(nc, n_r)] \times C_{perfGain} \times nc \tag{6.3}$$

$$U^{perfLoss}(nc, n_r, n_w) = [R(nc, n_r) - SLA] \times (C_{perfLoss} \times nc \times \frac{R(nc, n_r)}{SLA}) \tag{6.4}$$

$$U^{perf}(nc, n_r, n_w) = U^{perfGain}(nc, n_r, n_w) - U^{perfLoss}(nc, n_r, n_w) \tag{6.5}$$

where $[x]$ is an indication function which returns 0 when $x \leq 0$, and 1 otherwise, $U^{PerfGain}(nc, n_r, n_w)$ and $U^{PerfLoss}(nc, n_r, n_w)$ are the performance gain given the number of replicas allocated for the read and write set of the workload, respectively, $tx(nc, n_r)$ is an OAT model which correlates the number of clients nc and n_r to the query throughput, $R(nc, n_r)$ is the response time calculated according to the interactive response time law [21], and Z is the user average think time, $C_{perfGain}$ and $C_{perfLoss}$ are constants governing the weight of the performance gain and loss, respectively.

Finally, the overall utility is the sum of power and performance utilities i.e., $U(nc, n_r, n_w) = U^{pow}(nc, n_r, n_w) + U^{perf}(nc, n_r, n_w)$.

6.2 Utility-Based Provisioning Module

As before, the utility-based provisioning module uses the load prediction given by the workload predictor, and the utility computed by the OAT model to determine the replica configuration that optimizes the high level goal, in this case the utility. It actuates replica addition and removal accordingly.

Unlike performance-based provisioning, utility-based provisioning considers provisioning costs i.e., for power consumption, besides performance objectives. Therefore, when deciding the timing of a replica addition, the provisioning module needs to consider not only the estimated adaptation delay, but also the power consumption. As before, if a machine is provisioned too late, then it will incur a penalty due to SLA violations. However, in this case, if a machines

is provisioned too early, then it will incur a penalty due to the power consumption of the idle machine. Finally, the provisioning module needs to consider the uncertainty involved in predictions, i.e., the uncertainty inherent in the workload prediction and in the adaptation delay prediction.

We translate determining the best configuration among several candidate configurations into a problem of planning under uncertainty, using Markov decision processes. We use Markov decision processes to determine the best action to take at any given time. Each Markov state is the configuration that maximizes the utility, the transition probability is the combination of the prediction confidence and the estimated adaptation delay, and the trajectory is the series of actions to actuate server status change in order to maximize the expected utility.

Our utility-proactive scheme runs the provisioning algorithm periodically, at fixed intervals called *provisioning epochs*. In order to determine the optimal system trajectory, the provisioning module considers a *prediction look-ahead* of several epochs into the future. The prediction for each future epoch is associated with a confidence number p, such that $0 \leq p \leq 1$. The candidate configurations are the machine states which maximize the utility based on the predictions. We use dynamic programming [10] to determine the optimal trajectory. For the experiments, we use a prediction look-ahead of only two epochs, hence we consider a two-step trajectory.

Chapter 7

Baseline Reactive Provisioning Approach Used for Comparison

We use a baseline reactive scheme [36] 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.

Chapter 8

Experimental Results

In this chapter, after describing the experimental setup, we first show preliminary off-line experiments for load feature selection. Next, we show the benefit of our baseline proactive provisioning over the reactive approach when we use a performance-only objective. Then we show that this basic proactive provisioning scheme with on-line learning can adapt quickly to a change in workload mix. Finally, we augment the provisioning scheme with cost-awareness for reducing power and cooling costs, and we show that i) our proactive algorithm augmented with temperature awareness allows us to decrease the maximum temperature in the cluster, while still meeting the SLA. and ii) our utility-based proactive technique allows us to drive adaptations according to a given monetary utility encompassing both performance and power costs.

8.1 Experimental Setup

To evaluate our system, we use the TPC-W e-commerce benchmark [38] 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 [5] with PHP 4.0 [30] implementing the business logic and the MySQL 4.0 database server with InnoDB engine [27] 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 error threshold to 100*ms*. 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.

8.2 TPC-W E-Commerce Benchmark

The TPC-W benchmark from the Transaction Processing Council [38] 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.

8.3 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.

8.4 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/`

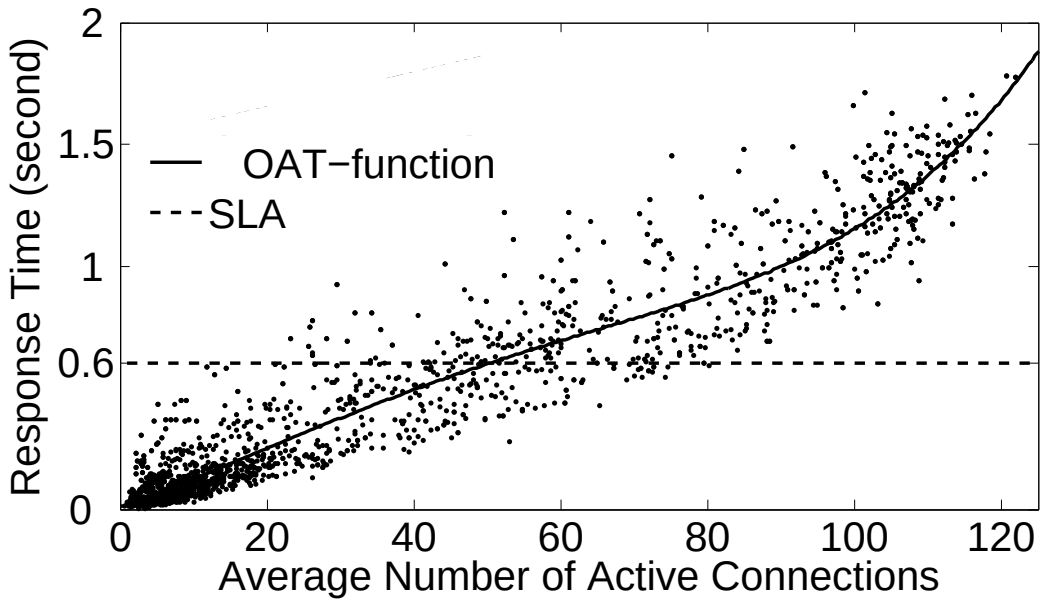


Figure 8.1: Correlation of number of active connections and response time

8.5 Preliminary Experiments

Table 8.1, shows an example of our feature selection process for SVM. As we can see from the table, 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 8.1 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. Hence, whenever we are interested in optimizing query response time as our only goal, we select the average number of active connections as the load feature for correlation tracking purposes in OAT. In a similar manner, we select the estimated number of clients as a feature highly correlated to CPU utilization in experiments where we are interested in the dual goal of optimizing both performance and power consumption.

Load metrics have non-linear relation with performance and power consumption. Fig-

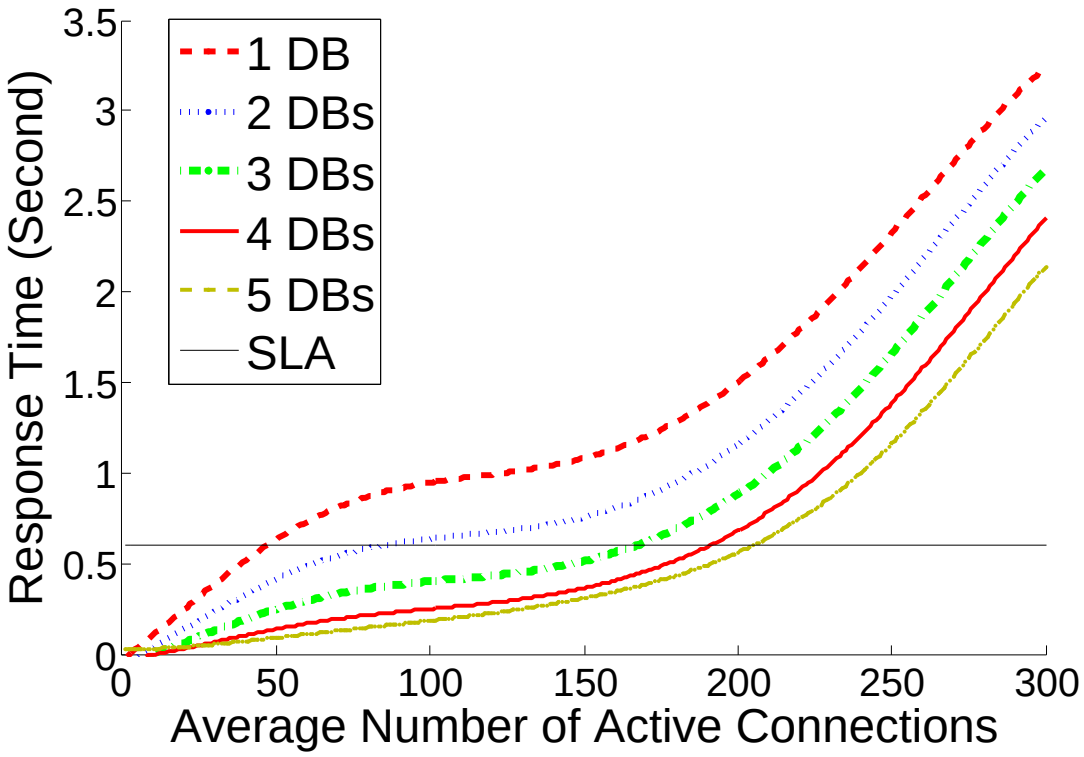


Figure 8.2: Correlation of number of active connections and response time for multiple replicas

ure 8.2 plots response time versus number of active connections for different number of databases. We see that the shape of the response time curve differs for each database configuration. Similarly as shown in Figure 8.3, CPU utilization, which we use to compute power consumption, has a non-linear relationship with the number of active connection. These results indicate that we need to use regression methods that can capture non-linear relationships, motivating our SVM approach.

8.6 Proactive versus Reactive Provisioning

In this section, we compare a proactive provisioning scheme targeted for performance only 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. We use an SLA corresponding to maintaining the query latency below 600 ms as the objective. As shown in Figure 8.4(a), we use

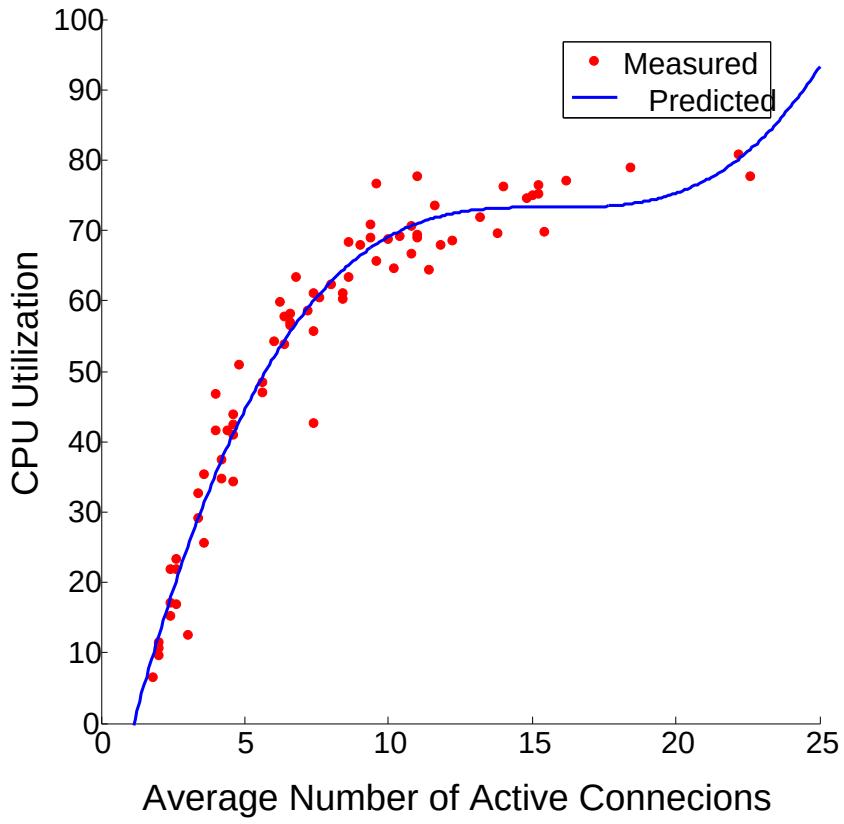


Figure 8.3: Correlation of number of active connections and CPU utilization

Table 8.1: 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

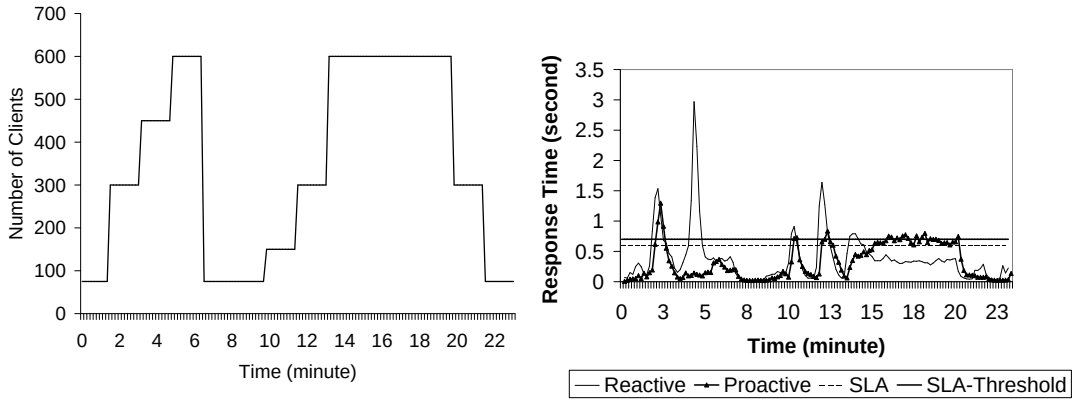
a load function with a variety of load patterns ranging from small steps (between 9-14 minute marks) to sharp drops (at the 20 minute mark) and sudden spikes (at the 14 minute mark). Figure 8.4(b) shows the response time of the reactive and proactive provisioning schemes. Figure 8.4(c) and Figure 8.4(d) show the number of replicas allocated by reactive and proactive provisioning algorithms, respectively.

Initially, i.e., roughly during the first 7 minutes of the experiment, 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, while our algorithm observes system metrics with different configurations and builds its initial set of samples. 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.

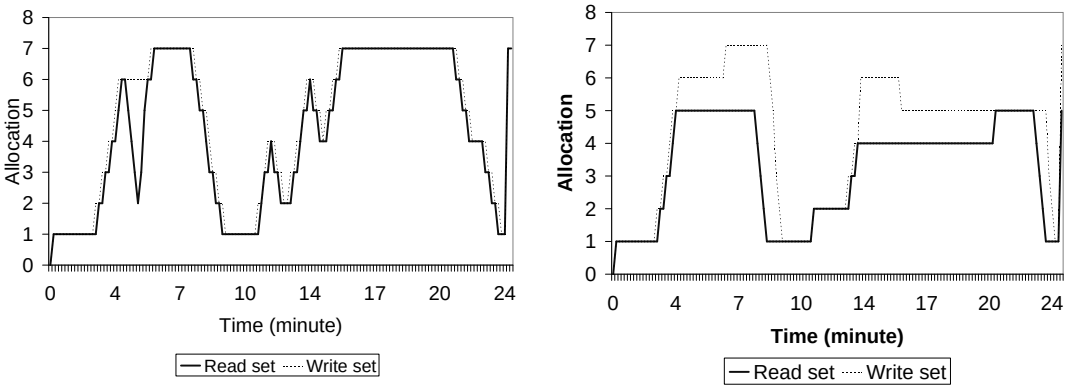
8.7 Adapting to Change of Workload Mix

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 8.5(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 8.5(c) shows that OAT can adapt well dynamically to the change of workload mix.



(a) Load Function

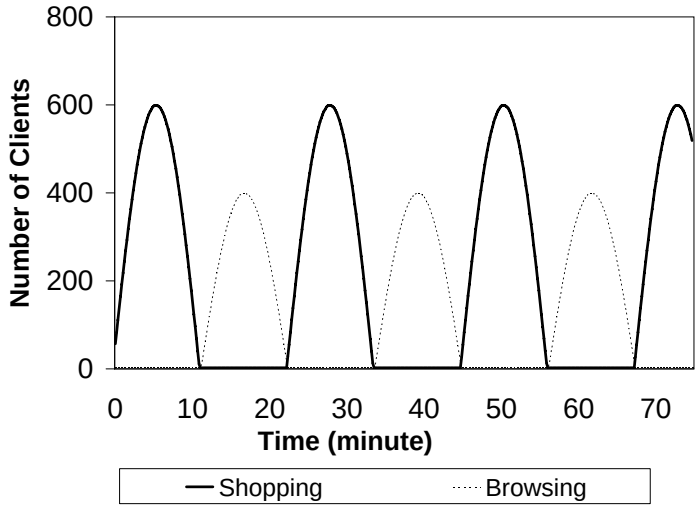
(b) Average latency



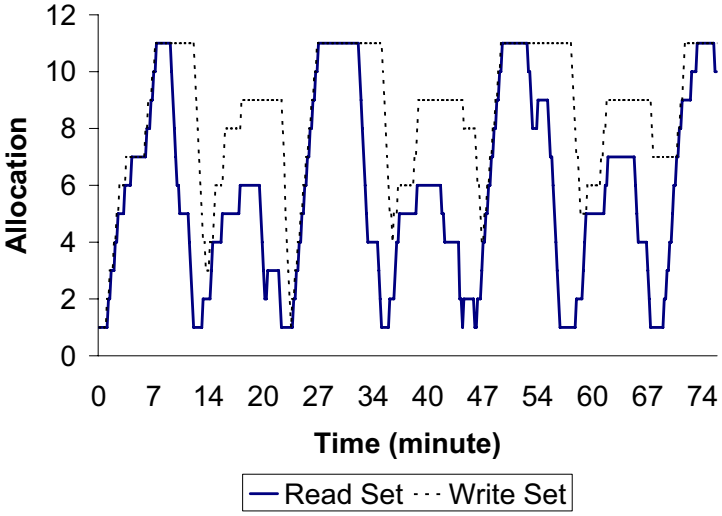
(c) Replica Allocation (*Reactive*)

(d) Replica Allocation (*Proactive*)

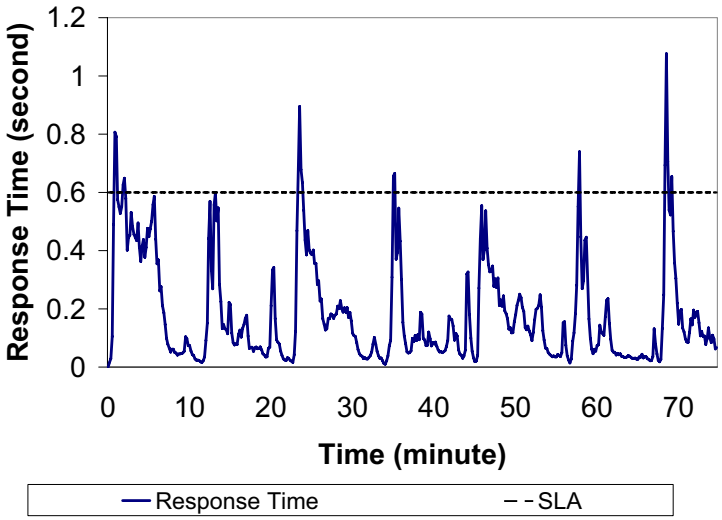
Figure 8.4: Comparison of the Proactive and Reactive provisioning schemes



(a) Load Function



(b) Replica Allocation



(c) Average Latency

Figure 8.5: Adapting to Changes in Workload Mix

Figure 8.5(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.

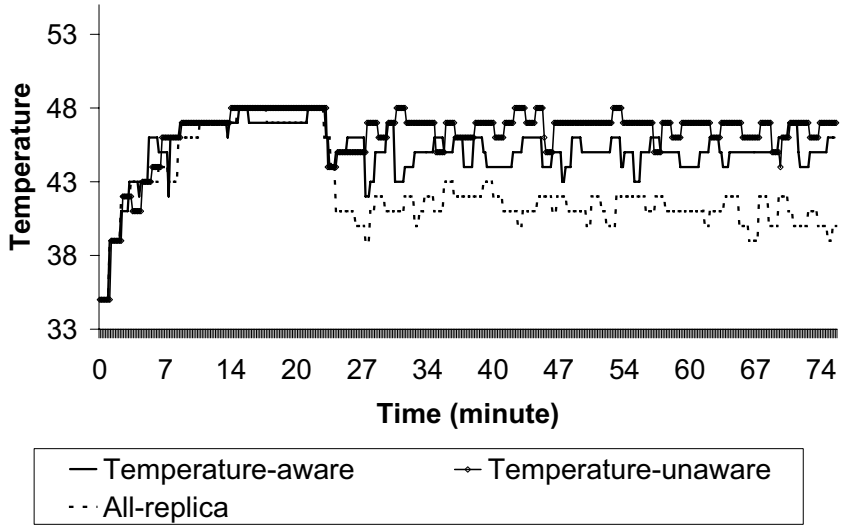
8.8 Evaluation of Temperature Aware Scheduling

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

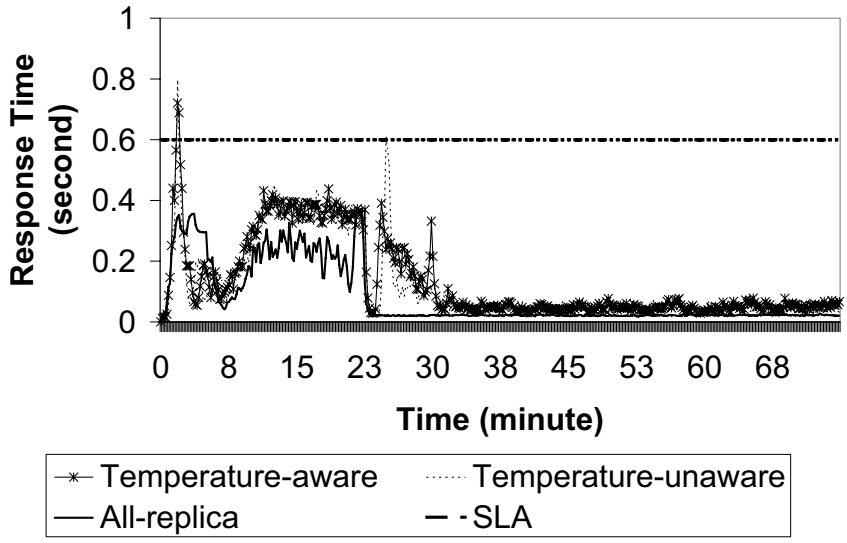
In the first scenario, shown in Figure 8.6, we compare the maximum temperature registered in the cluster for all schemes at the same resource usage. To obtain the same resource usage for all schemes (in this case 7 machines in the read set of the application and 11 machines in the write set), we disable 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 8.6(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 8.6(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



(a) Temperature



(b) Average Latency

Figure 8.6: Temperature Profile for Lightly Loaded Cluster

the cluster in a resource constrained scenario.

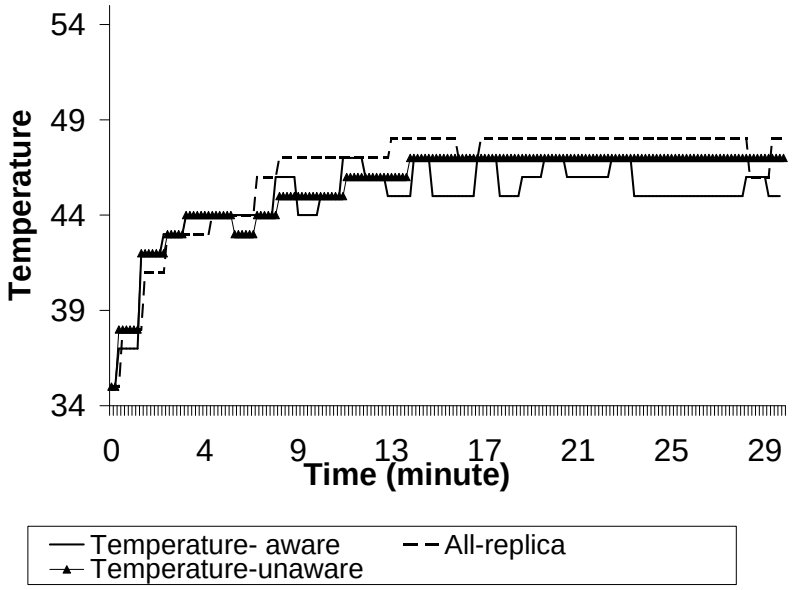
In this experiment we induce a flat load function with 500 emulated clients running the TPC-W browsing mix. The scheduler is constrained to use only 7 replicas with 5 of them in the read set of the application. As shown in Figure 8.7(b), the latency for all the three schemes is higher than before. The temperature profile depicted in Figure 8.7(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 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.

8.9 Evaluation of Utility-based Provisioning

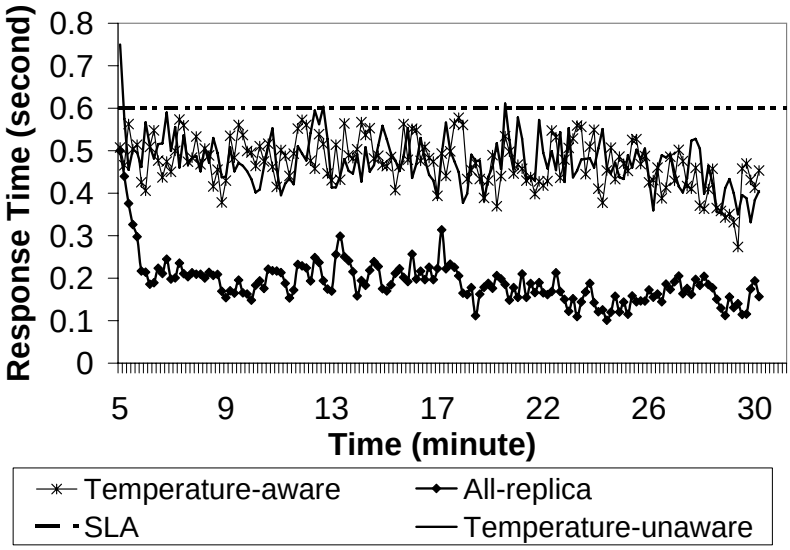
In this section, we evaluate our utility-based provisioning scheme.

8.9.1 Performance Vs. Power

We evaluate our cost-based dynamic provisioning scheme using a utility function combining power and performance. We show the adaptations and accuracy of our on-line adaptive tracking with two instances of utility function: *balanced utility* and *power-favored utility*. With *balanced utility*, the two objectives, power and performance have the same weight in the overall utility function; for example this would be the case when the penalty for SLA violations incurs similar costs to the operational costs due to power consumption. With *power-favored utility*, the utility function favors achieving power savings over meeting the performance SLA,

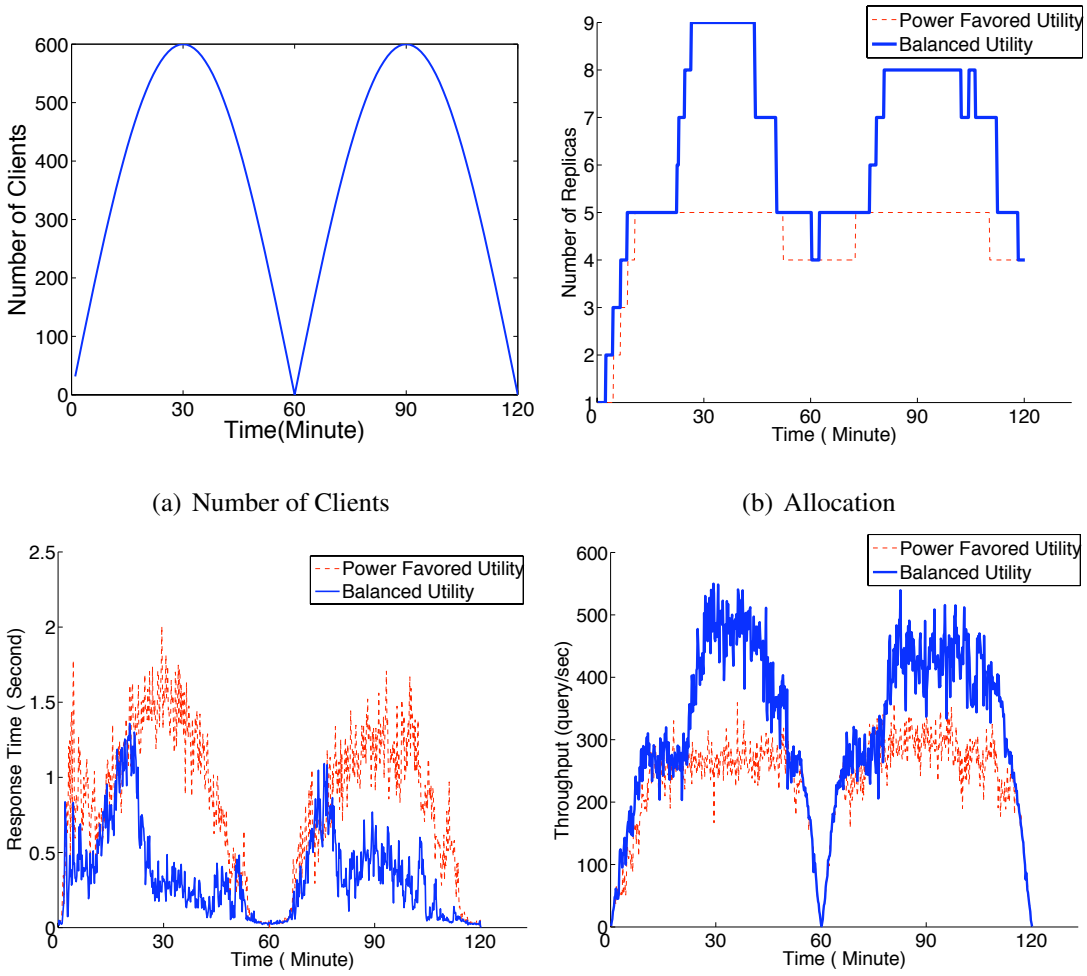


(a) Temperature



(b) Average Latency

Figure 8.7: Temperature Profile for Constrained Number of Replicas



(a) Number of Clients

(b) Allocation

(c) Average Latency

(d) Throughput

Figure 8.8: Power-favored vs. Performance-favored Utility

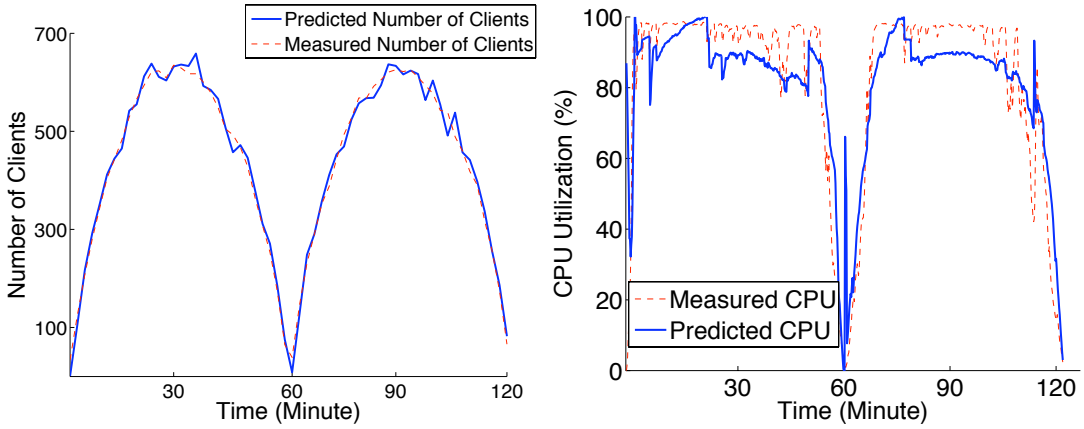
i.e., power costs are higher than SLA violation penalties.

Figure 8.8(a) shows the load function used in the experiments. Figure 8.8(b) shows that, in the case of power-favored utility, the provisioning module initially adapts by adding replicas during a load spike, but ultimately refrains from adding more replicas after the break-even point of 4 replicas is attained, regardless of performance. The effect of fewer allocations on the performance is shown in Figure 8.8(c) for latency and Figure 8.8(d) for throughput, where response time violates the SLA, and throughput remains constant despite the increase in the number of clients. In contrast, in the balanced utility case, we can see that the provisioning module adds more replicas during load spikes, achieving much shorter intervals of SLA violations compared to the balanced utility case. In both cases, the effectiveness of our algorithm in terms of the accuracy of approximating a given monetary utility is within the error threshold given. In the following we provide a detailed evaluation of the accuracy of our technique.

8.9.2 Accuracy of Prediction

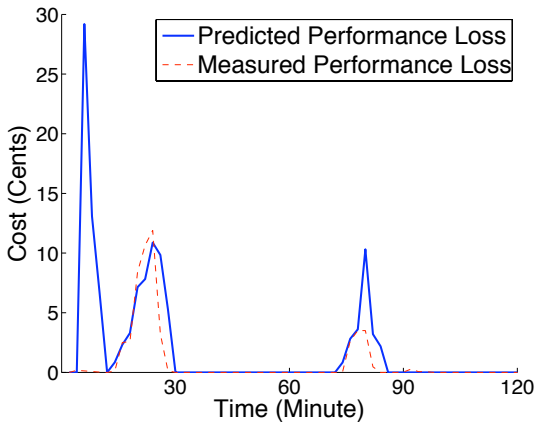
In this experiment, we evaluate the accuracy of the building blocks for our overall utility-based provisioning technique, i.e., the accuracy of the workload predictor, the accuracy of the OAT-based model, and the accuracy of the utility predictor.

Figure 8.9(a) shows the measured number of clients, and the number of clients as estimated by our workload predictor. We can see from the figure that the workload predictor has a high degree of accuracy. The accuracy of our OAT-based model is shown in Figure 8.9(b). The figure shows the measured CPU utilization versus the CPU utilization as estimated/predicted by our dynamic OAT model. The results show that OAT has a high degree of accuracy in correlating the CPU utilization to the number of clients under different replica configurations. Finally, Figures 8.9(c) and 8.9(d) show the measured and predicted utility in terms of its two components, performance and power loss, respectively. Again, we can see that except for the initial (brief) on-line training period, the predicted utility closely approximates the measured utility; hence the utility predictor component of our utility-based provisioning scheme exhibits

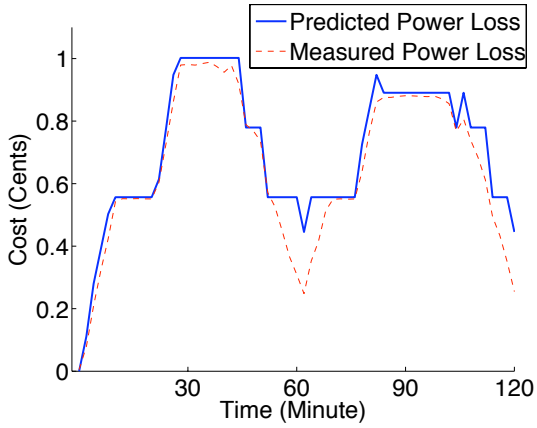


(a) Estimated Number of Clients

(b) CPU utilization



(c) Performance Loss



(d) Power Loss

Figure 8.9: Accuracy of Predictions

a high degree of accuracy.

In summary, the experiments shown demonstrate that our proactive utility-based provisioning scheme can predict the utility of various database tier configurations. As a result, our technique is able to actuate the addition and removal of database replicas in advance of predicted need, in order to maximize the defined monetary utility. This allows us to use a unifying goal for different performance/power cost scenarios.

Chapter 9

Related Work

In this dissertation, we investigate autonomic resource provisioning based on on-line learning, pursuing three orthogonal objectives: meeting a performance SLA, optimizing resource allocation, and improving the power consumption in a cluster of database servers.

While in our previous study [16], we consider temperature awareness as an enhancing feature of dynamic provisioning, power conservation is not explicitly taken into account when provisioning resources. In this dissertation, we develop a unifying framework for optimizing both performance and power consumption. We show how a utility-driven approach based on monetary costs can be used in conjunction with an on-line learning technique for this purpose.

Related work in the area of database provisioning based on learning techniques includes our previous KNN-based provisioning of database replicas[12]. This previous scheme requires extensive off-line training [12] of system states for each representative workload mix in order to build an accurate performance model. A similar table-driven provisioning approach [40] 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 [8, 41, 28] have also demonstrated good accu-

racy 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. [13] and Zhang et al. [43] 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 power-aware enhancements for a dynamic provisioning scheme draw on previous work describing simulations based on thermodynamic concepts [26]. 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 [11, 29] 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.

Pinheiro et al. [31] proposes a technique that leverages the redundancy in storage systems to conserve disk energy. They evaluate their work based on simulation. The proposed model is different from our work in that we deal with limitation of keeping replicas consistent in enforcing policies for reduction of power consumption.

Chapter 10

Conclusions

In this dissertation, 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 application-level metrics, ii) allocating database replicas in advance of need based on the dynamic performance model and load prediction, and iii) incorporating cost-awareness for power and cooling costs into pro-active provisioning.

Our proactive 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 enhance our proactive provisioning technique with two independent cost-aware features as follows: i) temperature-awareness allows us to shift the load from hot-spot nodes to newly allocated low-temperature nodes, thus reducing cooling costs and ii) monetary cost-awareness allows us to provision resources only as long as profitable for the provider, given any performance/power cost balance.

Our experimental evaluation based on the TPC-W e-commerce benchmark shows that the two cost-aware provisioning techniques are effective in meeting their respective objectives. The 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 using 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.

The utility-based proactive technique is accurate in its load, resource usage and utility predictions, hence is able to optimize its actuation of resources according to any given utility function. As a result, our utility-based provisioning technique can predict and adapt towards database configurations that maximize the monetary profit of the service provider regardless of power and performance costs in the data center.

Bibliography

[1] C. Amza, E. Cecchet, A. Chanda, A. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel. Specification and implementation of dynamic web site benchmarks. In *5th IEEE Workshop on Workload Characterization*, November 2002.

[2] C. Amza, A. Cox, and W. Zwaenepoel. Distributed versioning: Consistent replication for scaling back-end databases of dynamic content web sites. In *4th ACM/IFIP/Usenix International Middleware Conference*, pages 282–304, June 2003.

[3] C. Amza, A. L. Cox, and W. Zwaenepoel. A comparative evaluation of transparent scaling techniques for dynamic content servers. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, pages 230–241, Washington, DC, USA, 2005. IEEE Computer Society.

[4] Cristiana Amza, Alan L. Cox, and Willy Zwaenepoel. Conflict-aware scheduling for dynamic content applications. In *USITS'03: Proceedings of the 4th conference on USENIX Symposium on Internet Technologies and Systems*, pages 6–6, Berkeley, CA, USA, 2003. USENIX Association.

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

[6] Martin F. Arlitt and Carey L. Williamson. Internet web servers: workload characterization and performance implications. *IEEE/ACM Transactions on Networking (TON)*, 5(5):631–645, 1997.

[7] Yuliy Baryshnikov, Ed Coffman, Guillaume Pierre, Dan Rubenstein, Mark Squillante, and Teddy 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.

[8] Mohamed N. Bennani and Daniel A. Menasce. Resource allocation for autonomic data centers using analytic performance models. In *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, pages 229–240, Washington, DC, USA, 2005. IEEE Computer Society.

[9] P.A. Bernstein, V. Hadzilacos, and N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley, Reading, Massachusetts, 1987.

[10] Craig Boutilier. Decision making under uncertainty: Operations research meets ai (again). In *Proceedings of the Seventeenth National Conference on Artificial Intelligence and Twelfth Conference on Innovative Applications of Artificial Intelligence*, pages 1145–1150. AAAI Press / The MIT Press, 2000.

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

[12] Jin Chen, Gokul Soundararajan, and Cristiana 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.

[13] Ira Cohen, Moises Goldszmidt, Terence Kelly, Julie Symons, and Jeffrey S. Chase. Correlating instrumentation data to system states: a building block for automated diagnosis and control. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 16–16, Berkeley, CA, USA, 2004. USENIX Association.

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

[15] Harris Drucker, Christopher J. C. Burges, Linda Kaufman, Alex J. Smola, and Vladimir Vapnik. Support vector regression machines. In *Neural Information Processing Systems (NIPS)*, pages 155–161, 1996.

[16] Saeed Ghanbari, Gokul Soundararajan, Jin Chen, and Cristiana Amza. Adaptive learning of metric correlations for temperature-aware database provisioning. In *ICAC '07: Proceedings of the Fourth International Conference on Autonomic Computing*, page 26, Washington, DC, USA, 2007. IEEE Computer Society.

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

[18] IBM. Autonomic Computing Manifesto. http://www.research.ibm.com/ autonomic/manifesto, 2003.

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

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

[21] Raj Jain. *The Art of Computer Systems Performance Analysis: Techniques for Experimental Design, Measurement, Simulation and Modelling* . John Wiley & Sons, New York, 1991.

[22] A. Karve, T. Kimbrel, G. Pacifici, M. Spreitzer, M. Steinder, M. Sviridenko, and A. Tantawi. Dynamic placement for clustered web applications. In *WWW '06: Pro-*

ceedings of the 15th international conference on World Wide Web, pages 595–604, New York, NY, USA, 2006. ACM.

[23] Ed Lassettre, D. W. Coleman, Yixin Diao, Steve Froehlich, Joseph L. Hellerstein, L. Hsiung, T. Mummert, M. Raghavachari, G. Parker, L. Russell, Maheswaran Surendra, V. Tseng, N. Wadia, and P. Ye. Dynamic surge protection: An approach to handling unexpected workload surges with resource actions that have lead times. In *First Workshop on Algorithms and Architectures for Self-Managing Systems*, pages 82–92, 2003.

[24] Daniel A. Menasce;, Daniel Barbara;, and Ronald Dodge. Preserving qos of e-commerce sites through self-tuning: a performance model approach. In *EC '01: Proceedings of the 3rd ACM conference on Electronic Commerce*, pages 224–234, New York, NY, USA, 2001. ACM Press.

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

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

[27] MySQL. `http://www.mysql.com`.

[28] Dushyanth Narayanan, Eno Thereska, and Anastassia 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.

[29] Steven Osman, Dinesh Subhraveti, Gong Su, and Jason Nieh. The design and implementation of zap: a system for migrating computing environments. *Proceedings of 5th*

USENIX Symposium on Operating Systems Design and Implementation, 36(SI):361–376, 2002.

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

[31] Eduardo Pinheiro, Ricardo Bianchini, and Cezary Dubnicki. Exploiting redundancy to conserve energy in storage systems. *ACM SIGMETRICS Performance Evaluation Review*, 34(1):15–26, 2006.

[32] S. Ranjan, J. Rolia, H. Fu, and E. Knightly. QoS-Driven Server Migration for Internet Data Centers. In *10th International Workshop on Quality of Service*, May 2002.

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

[34] Slashdot: Handling the Loads on 9/11. `http://slashdot.org`.

[35] Gokul Soundararajan and Cristiana Amza. Reactive provisioning of backend databases in shared dynamic content server clusters. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 1(2):151–188, 2006.

[36] Gokul Soundararajan, Cristiana Amza, and Ashvin Goel. Database replication policies for dynamic content applications. In *EuroSys'06: Proceedings of the EuroSys 2006 Conference*, pages 89–102. ACM, 2006.

[37] Gerald Tesauro, William E. Walsh, and Jeffrey O. Kephart. Utility-function-driven resource allocation in autonomic systems. In *ICAC '05: Proceedings of the Second International Conference on Automatic Computing*, pages 342–343, Washington, DC, USA, 2005. IEEE Computer Society.

[38] Transaction Processing Performance Council. TPC benchmark B standard specification, August 1990.

[39] Andrew Trossman. Tools and techniqes for data centre automation. In *CASCON'06: Third Annual Workshop on Self-Optimizing Systems*. IBM Corporation, Tivoli, 2006.

[40] William E. Walsh, Gerald Tesauro, Jeffrey O. Kephart, and Rajarshi Das. Utility functions in autonomic systems. In *Proceedings of the 1st International Conference on Autonomic Computing (ICAC)*, 2004.

[41] Murray Woodside, Tao Zheng, and Marin 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.

[42] Shuqing Wu and Bettina Kemme. Postgres-r(si): Combining replica control with concurrency control based on snapshot isolation. In *ICDE '05: Proceedings of the 21st International Conference on Data Engineering*, pages 422–433, Washington, DC, USA, 2005. IEEE Computer Society.

[43] Steve Zhang, Ira Cohen, Julie Symons, and Armando Fox. Ensembles of models for automated diagnosis of system performance problems. In *DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks*, pages 644–653, Washington, DC, USA, 2005. IEEE Computer Society.