Automated Control for Elastic n-Tier Workloads based on Empirical Modeling

Simon Malkowski
CERCS,
Georgia Institute of Technology
266 Ferst Drive
Atlanta, GA 30332, USA
simon.malkowski@cc.gatech.edu

Markus Hedwig
Chair of Information Systems,
Albert-Ludwigs-University
Platz der Alten Synagoge
79805 Freiburg, Germany
markus.hedwig@is.uni-freiburg.de

Jack Li
CERCS,
Georgia Institute of Technology
266 Ferst Drive
Atlanta, GA 30332, USA
jack.li@cc.gatech.edu

Calton Pu
CERCS,
Georgia Institute of Technology
266 Ferst Drive
Atlanta, GA 30332, USA
calton.pu@cc.gatech.edu

Dirk Neumann
Chair of Information Systems,
Albert-Ludwigs-University
Platz der Alten Synagoge
79805 Freiburg, Germany
dirk.neumann@is.uni-freiburg.de

ABSTRACT

Elastic n-tier applications have non-stationary workloads that require adaptive control of resources allocated to them. This presents not only an opportunity in pay-as-you-use clouds, but also a challenge to dynamically allocate virtual machines appropriately. Previous approaches based on control theory, queuing networks, and machine learning work well for some situations, but each model has its own limitations due to inaccuracies in performance prediction. In this paper we propose a multi-model controller, which integrates adaptation decisions from several models, choosing the best. The focus of our work is an *empirical model*, based on detailed measurement data from previous application runs. The main advantage of the empirical model is that it returns high quality performance predictions based on measured data. For new application scenarios, we use other models or heuristics as a starting point, and all performance data are continuously incorporated into the empirical model's knowledge base. Using a prototype implementation of the multi-model controller, a cloud testbed, and an n-tier benchmark (RUBBoS), we evaluated and validated the advantages of the empirical model. For example, measured data show that it is more effective to add two nodes as a group, one for each tier, when two tiers approach saturation simultaneously.

Categories and Subject Descriptors

C.2.4 [**Computer-Communication Networks**]: Distributed Systems—*Distributed applications*; C.4 [**Performance of Systems**]: Modeling techniques; I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search—*Control theory*

General Terms

Experimentation, Management, Measurement, Performance.

Keywords

Adaptation engine, Automated control, Cloud, Empirical modeling, Elastic system, n-Tier application, Workload.

1. INTRODUCTION

Efficiency is a non-trivial challenge when managing the performance of web-facing applications, such as e-commerce websites and social networks, because their workloads are characterized by sudden and periodic variations. In fact, these non-stationary workloads have peak loads several times the sustained load and are particularly difficult to forecast accurately [1, 2, 6]. If these *elastic* applications are deployed in a dedicated data center, the service provider will face a challenging trade-off. On the one hand, satisfying QoS objectives such as short response time at peak loads will keep a large number of nodes at low utilization levels most of the time. On the other hand, provisioning mainly for the sustained load by keeping a relatively high utilization of a small number of nodes will lead to saturation and loss of business during peak loads.

Cloud environments have been touted as good solution for elastic applications because independent peak loads allow sharing of the same nodes using techniques such as consolidation; however, the relative immaturity of cloud technology renders many important technical challenges not (yet) addressed. For instance, while achieving an economical sharing of cloud resources at sustained loads *and* satisfying QoS objectives at peak loads is clearly of great interest for both cloud providers and cloud users, the necessary management

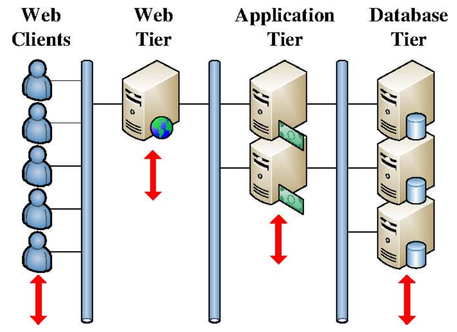


Figure 1: Elastic four-tier application.

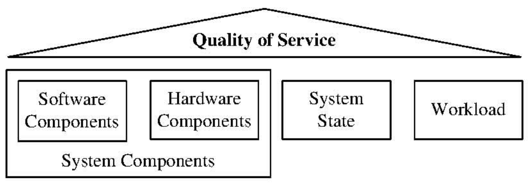


Figure 2: Major quality of service determinants.

automation is still an open (commercial and academic) research question. In fact, to meet this challenge, automation of adaptive resource provisioning for elastic applications is one of the major topics of interest for the autonomic computing community [12, 18, 19, 25, 27].

This paper focuses on elastic n-tier applications that are deployed on Infrastructure as a Service (IaaS) cloud environments such as Amazon EC2. A three-tiered application example with web clients, web tier, application tier, and database tier is shown in Figure 1. Such n-tier applications are common because of their modular structure and potential for performance scalability by adding more servers. However, due to the dependencies among the servers, it is non-trivial to scale n-tier applications in a cloud environment [12, 14, 15, 25]. Moreover, Figure 2 illustrates other factors that may significantly affect the system QoS, including hardware and software components (e.g., type of storage), system state (e.g., data size and distribution), and workload (e.g., workload level and mix). We call these factors QoS *determinants*. Due to the dependencies among servers, potentially inaccessible QoS determinants, and uncontrollable environment changes, continuous performance modeling of elastic n-tier applications is a complex non-linear analysis problem with multiple levels. Our approach extends automated control (e.g., [12, 25]) through detailed measurement storage that captures the current dependencies among components and layers empirically in a direct mapping of QoS to workload and tunable system knobs. This facilitates automated and incremental modeling of complex performance characteristics at high accuracy and ensures continuous model evolution over time.

The conceptual contribution of this paper is a novel Integrative Adaptation Engine (IAE) architecture with a multi-model controller (Figure 3). The controller monitors an elastic n-tier application as workloads and QoS determinants vary, and decides on an appropriate allocation of Virtual Machine (VM) instances per tier, which we refer to as *configuration*. We call our controller *multi-model* because it integrates several different models to determine "the best" adaptation action; e.g., in terms of lowest adaptation costs [11]. The major innovation of the multi-model controller is its *empirical model* that is designed as an interface to the Operational Data Store (ODS) in Figure 3. Based on queries that are triggered by the controller logic, the ODS searches through a knowledge base of previously measured operational performance data for similar configurations and workloads. If a recent match is found, the quality of the controller's adaptation decisions should be very high (at least as high as with other a priori instantiated models) since the system performance should match the previously measured results. An important feature of the empirical model is its ability to gradually increase its prediction precision and accuracy by accumulating new measurement data over time. The main advantage of the multi-model controller is its inherent extensibility with regard to other performance prediction models (ranging from simple heuristics to sophisticated layered queuing networks) through a unified meta-model interface.

The technical contribution of this paper is an experimental evaluation of the empirical model approach, using a prototype implementation of the IAE. The evaluation demonstrates the advantages of the empirical model running an elastic n-tier application workload. The IAE prototype delivers high quality adaptation decisions based on three important elements: (1) multi-model control, (2) workload forecast, and (3) continuous learning. First, the multi-model controller maintains QoS through adaptation decisions from two models: the empirical model and an additive horizontal scale model [12] that allocates nodes one-by-one according to CPU utilization. If a match is found in the ODS, the empirical model decision is adopted; otherwise, the additive horizontal scale model is used. Second, a workload forecast model [8] predicts short-term future workload through Fourier Transformations to mitigate seasonal workload fluctuations and adaptation lead-times. Third, a learning module captures the running application's performance data and stores them into the ODS continuously to augment the model's knowledge base. The empirical model's knowledge base can initially be seeded with a set of measurements from automated experiments [13, 15].

Our experimental evaluation shows that the empirical model can deliver more effective adaptation actions, compared to models such as the additive horizontal scale model. For example, our measurements show that the intuitive strategy of adding/subtracting nodes one-by-one becomes sub-optimal when two mutually-dependent tiers are approaching saturation simultaneously. Adding a node to one tier requires the immediate addition of a node to the dependent tier to prevent an overall performance degradation effect. Consequently, it may be more cost effective to add those nodes as a group adaptation, instead of doing it one-by-one.

This work builds on the foundations and lessons learned from a number of our previous papers. Prior to this work, we have explored performance analysis and control from the perspectives of control theory [27], queuing theory [10, 11], automated learning [26], and empirical analysis [7, 8]. In parallel, we have experimentally analyzed cloud system scalability with a particular focus on n-tier applications [15]. Based on our findings, we have identified *multi-bottleneck* phenomena outside of classical modeling assumptions [14], which led

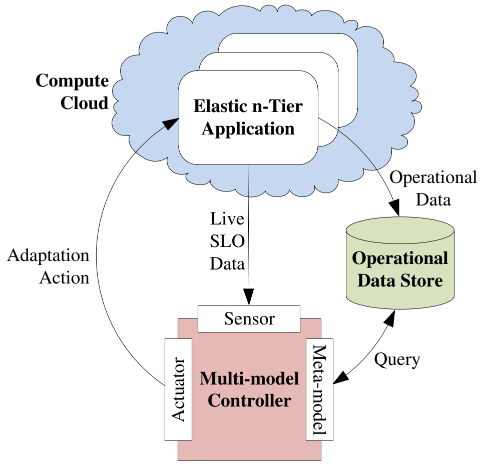


Figure 3: Integrative Adaptation Engine architecture.

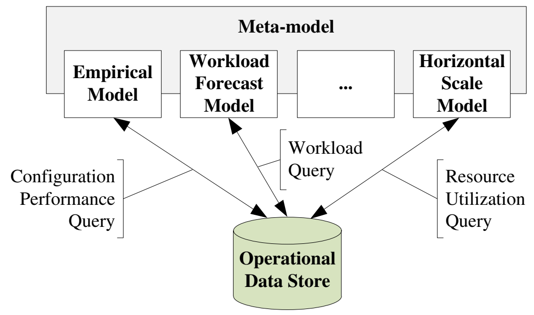


Figure 4: Extensible meta-model that implements the EM, the WFM, and the HSM with ODS access.

to the development of an empirical configuration planning tool [13].

The rest of this paper is organized as follows. Section 2 introduces our Integrative Adaptation Engine architecture. In Section 3, we describe the elastic application testbed, provide IAE implementation details, and discuss our experimental results. Section 4 further contextualizes our work and provides an overview of related approaches. We conclude the paper in Section 5 with a brief summary and discussion of our findings.

2. ADAPTATION ARCHITECTURE

The three main components of the IAE architecture are the elastic application system, the multi-model controller, and the Operational Data Store (ODS). In this context, the elastic application system comprises both the compute cloud (hardware-side) and the elastic n-tier application (software-side). Figure 3 provides a high-level overview of the topology and communication between these components.

The elastic n-tier application is hosted on VM instances that are obtained from the cloud infrastructure provider. VM instances are available on a commodity pricing basis (i.e., pay-as-you-use) with fine-grained rental periods. The guest application can be assumed to be a web-facing application with multiple tiers, which serve a web-interaction workload generated by an elastic number of independent clients. During operation, an n-tier application system exposes various hooks that allow the recording of rich monitoring data. Typically, the quality of the provided service (i.e., the user experience) is characterized in the form of Service Level Objectives (SLOs), which are combined into Service Level Agreements (SLAs). In our architecture the percentage of satisfied SLOs is directly reported to the adaptation controller's sensor as a live data stream.

The persistent storage module for all available operational data such as resource utilizations, number of VM instances per tier, and interaction-specific throughput is the Operational Data Store (ODS). The ODS enables the IAE to make adaptation decisions based on previously observed application system behavior, and is the second of the three main IAE components. The third is the multi-model controller, which interacts with the outside world through three interfaces: the sensor, the meta-model, and the actuator. In particular, the sensor interface is responsible for triggering the adaptation decision workflow in response to a live SLO input. During the decision-making process the adaptation controller uses the meta-model interface to determine the "state of the outside world". The meta-model provides access to all previously recorded operational data in the ODS (see the following sections). If the controller determines that the elastic application system can be optimized, an appropriate adaptation action is triggered through the actuator interface. Typically, adaptation actions have to be communicated to both the cloud provider's API as well as the elastic n-tier application's API.

From a more general perspective, the main data-flow in Figure 3 constitutes an infinite cycle. This design facilitates the continuous extension of the state-space that is known to the empirical model. Conceptually, this design provides the potential for high modeling accuracy resulting from dynamic data evolution over time, which corresponds to continuous model evolution.

2.1 Meta-model

The meta-model is the the multi-model's interface that enables the adaptation controller to access previously observed operational data in the ODS. As shown in Figure 4, the meta-model is extensible and can implement a number of sub-models. For this paper, we have restricted the meta-model to three sub-models: the Empirical Model (EM), the Workload Forecast Model (WFM), and the Horizontal Scale Model (HSM). The analysis of more complex hybrid-model solutions are beyond the scope of this paper and are left as future work.

2.1.1 Horizontal Scale Model

The purpose of the HSM is to query resource utilization values in order to determine which tier is over-utilized (or under-utilized). As previously mentioned, this type of scale decision analysis was employed by Lim et al. in their Horizontal Scale Controller (HSC) [12]. Following Lim et al., the HSM classifies the tier with the highest (or lowest) CPU-utilization average in the past measurement interval as over-utilized (or under-utilized). In our approach, the HSM serves as an alternative decision method to the EM. As detailed in Section 2.2, this alternative decision method is necessary in case the queried state-space is not (sufficiently) known to the EM. In other words, the HSM compensates the main drawback of the EM, which is the dependence on previously recorded data. In combination the two models facilitate both efficient adaptation to previously encountered scenarios and adaptation to previously unseen scenarios.

2.1.2 Empirical Model

The EM queries the ODS for performance achieved by distinct configurations based on throughput vectors. Correspondingly, the performance data in the ODS are organized in a throughput vector space, which is multidimensional, application-specific, and cloud-specific. Each vector $\vec{x}$ in the ODS summarizes a set of interaction-type throughput rates for a short time interval. Each interaction corresponds to a distinct dimension; e.g., $\vec{x} = (x_1, x_2, \ldots, x_n)'$ where x_i is the throughput of interaction-type i during one specific measurement interval. This data organization enables the EM to query the recorded data based on an anticipated throughput rate vector $\vec{y}$ that can be derived from the currently experienced workload level and a short-term workload prediction (see Section 2.1.3). The goal of a query to the ODS is to retrieve "suitable" configurations, which are capable of sustaining the anticipated throughput $\vec{y}$. In this context, the suitability of a configuration can be defined as an insignificant (i.e., short) vector space distance between the predicted throughput $\vec{y}$ and the previously recorded throughput $\vec{x}$.

The vector space distance for our ODS is formulated on the basis of the well-known Euclidean norm. We transform the classic Euclidean distance in order to include additional domain-specific constraints. Correspondingly, we define our own distance function (1) as a summation of function f.

$$dist(\vec{x}, \vec{y}) = ||\vec{x} - \vec{y}|| = \sqrt{\sum_{i=1}^{n} f(x_i, y_i)} \qquad (1)$$

Function f is defined in Equation (2) and uses the Euclidean metric if the recorded throughput is less than or equal to the query throughput. The Euclidean metric value is multiplied by a weighting parameter $\alpha \in [0, 1]$ in all other cases.

$$f(x_i, y_i) = \begin{cases} (x_i - y_i)^2 & \text{if } x_i \leq y_i \\ \alpha * (x_i - y_i)^2 & \text{otherwise} \end{cases} \qquad (2)$$

The intuition behind this definition is to retrieve configurations that are capable of sustaining a throughput that is greater than or equal to the query throughput. Therefore, the distance should mainly increase through throughput deviations that are negative, which violates the desired performance. In the most intuitive case, weighting parameter α is equal to zero. However, it may also be desirable to extend the state-space exploration phase of the IAE. In these cases, a higher weight results in a more frequent use of alternative sub-models, which leads to a higher state-space coverage.

The EM queries to the ODS can be designed according to the following pseudo-SQL based on the scalar stored function `DIST(@x_vector, @y_vector)`, which implements the aforementioned distance metric.

```
SELECT
    MAX(time_stamp) AS last_used, AVG(sla) AS sla,
    config AS configuration_setting, cost,
    COUNT(id) AS support
FROM Operational_Data
WHERE
    DIST(@x_vector, @y_vector) < @threshold
GROUP BY
    configuration_setting
ORDER BY
    cost, last_used DESC;
```

The scalar `@threshold` is substituted by an appropriate distance threshold. The choice of this threshold determines the domain-specific selectivity of the EM with regard to judging deviations from previously recorded throughput values. Higher selectivity increases the chances of relying on alternative sub-models (e.g., the HSM) in the meta-model. Further selectivity constraints can be formulated on SLA-satisfaction and support in the result set. While the latter ensures that there is sufficient evidence of the suitability for a given configuration, the `sla` column ensures that the desired QoS was actually met. The retrieved tuples from the ODS are grouped by distinct configurations (e.g., cardinality per tier) and are ordered by the configuration cost. In horizontal scaling scenarios with uniform VM types, the cost is proportional to the total number of VM instances. The secondary ordering attribute `last_used` corresponds to the most recently observed configuration. This facilitates the continuous evolution of the performance model if changes occur to the underlying performance determinants. In other words, the model gives preference to the most recently recorded data, which can be regarded as an intuitive form of data aging in a learning module.

2.1.3 Workload Forecast Model

Although workload is one of the critical QoS determinants (Figure 2), current automated adaptation systems (e.g., [12]) are often purely reactive. This may cause inefficient adaptation if the workload level is highly volatile [25] or even oscillating system size due to periodic workload fluctuations. This risk is compounded by adaptation lead-times between the request of additional resources and their availability. Therefore, it is necessary, under the aforementioned conditions, to provide the multi-model controller with a predictive understanding of the application's workload process. We approach this challenge by designing a WFM, which allows the controller to pro-actively adapt the system before actually facing a specific workload scenario.

Typical elastic applications are characterized by a large number of users that send independent requests, and by a workload that may seasonally vary by more than one order of magnitude in a single day [6]. Therefore, we have previously designed a forecast mechanism based on Fourier Transformation (FT) for this specific class of workloads [8]. More concretely, the WFM queries the ODS for historic workload data whereby the query directly aggregates the high-resolution data into a second-level time series. The main seasonal effects are determined based on FT. As FT decomposes the process into a set of trigonometric functions, each of the main factors of influence can be extrapolated. The near future behavior of the process is predicted by overlaying all significant extrapolations and determining the relative change. The predicted workload-level change is then multiplied with the current interaction-specific throughput to estimate the short-term workload development. Please refer to the cited reference for a comprehensive description of this previously published result.

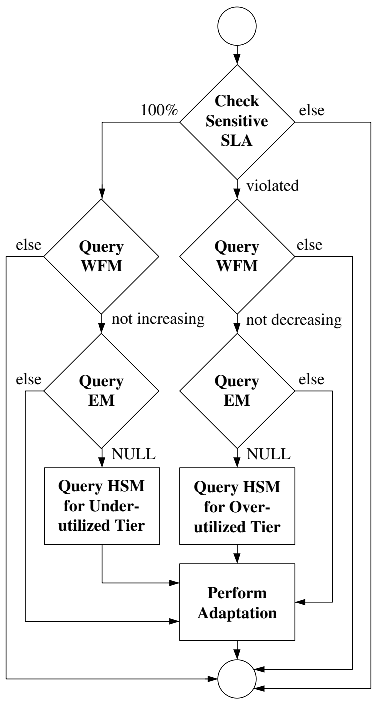


Figure 5: Decision-making workflow inside the multi-model controller.

2.2 Adaptation Workflow

The decision-making process inside the multi-model controller can be represented as a workflow graph with multiple sequential and parallel decision blocks. The more sub-models that are included in the meta-model, the more rule-based decision logic has to be added to the workflow. In the case of our initial prototype, the decision logic has to manage the dataflow from the EM, the WFM, and the HSM. The corresponding workflow graph in Figure 5 has two main execution paths. While the path on the left leads to a potential scale-down adaptation, the right path leads to a potential scale-up adaptation. The triggering of these adaptations is based on *sensitive SLA*, which sets higher QoS requirements than *normal SLA*. The definition of a sensitive metric allows initiating a scale-up adaptation before the normal SLA (i.e., the actual QoS metric) is violated, and it similarly allows initiating a scale-down adaptation only if the sensitive SLA is fully satisfied. In contrast to our approach, previous control approaches (e.g., [12]) typically only monitor QoS indirectly by assuming strong correlation between QoS and a representative resource utilization metric such as CPU utilization.

The two main execution paths in Figure 5 are analogous for both scale-up and scale-down. First, the adaptation workflow proceeds only if either of the two aforementioned sensitive SLA requirements is met. Second, a short-term workload prediction is obtained. Third, the ODS is queried in order to obtain an empirically founded adaptation action. Fourth, if the ODS query resulted in a NULL-result, the HSM is queried to make an adaptation decision exclusively based on CPU utilization. Finally, the previously determined adaptation action is initiated through the actuator interface.

3. EXPERIMENTAL EVALUATION

3.1 Elastic Application Testbed

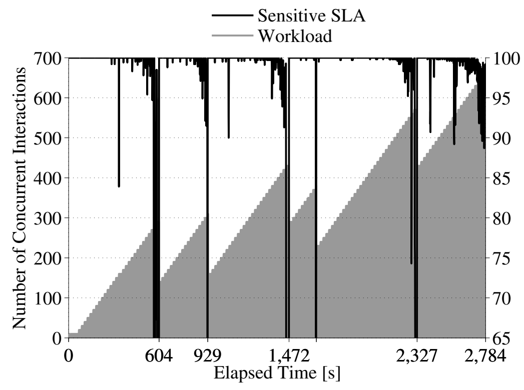
RUBBoS [22] is an n-tier e-commerce application modeled on bulletin board news sites similar to Slashdot. We have used the benchmark implementation with four tiers (i.e., client emulator, web server, application server, and database server). In general, this application places high load on the backend. The generated workload consists of 24 different interactions (involving all tiers) such as register user, view story, and post comments. We ported the Java implementation of RUBBoS to the Microsoft .NET environment. We modified the benchmark to run six native RUBBoS interactions, and we implemented a seventh interaction that we call the Business Logic Request (BLR). The BLR has the properties of a typical CPU-bound application-server-intensive interaction. More concretely, complex (CPU-intensive) operations are performed in the application tier based on simple database queries with relatively small result sets. In essence, the resulting benchmark presents itself as a well-balanced system that distributes the load evenly among all three tiers[1], which is the main target environment of this paper. The .NET code was deployed in enterprise class Microsoft environment. The frontend and the application server tier were deployed on Microsoft Internet Information Services (IIS) servers. As a backend, Microsoft SQL Server was used with the schema and large dataset provided by the RUBBoS benchmark. We modified the RUBBoS benchmark workload generator to use an open model with a freely manageable number of concurrent user interactions (i.e., no client thread sleep time) [6]. The interactions are chosen based on an a priori specified probability vector. All benchmark application components were deployed on Microsoft Windows Server 2003 (SP2).

We used a custom local cluster as our cloud infrastructure. We implemented scripts that allow us to obtain and release virtual resource similar to a real IaaS cloud environment. The test cluster was built using hardware virtualized Xen 3.4.3 VM instances. The underlying operating system was Red Hat Enterprise Linux 5 (2.6.18-194.11.1.el5xen kernel) on 12 hardware nodes with 8GB of memory and an Intel Core2 QuadCPU (Q9650) with 3.00GHz each. All VMs were created with one VCPU and 2GB of memory into a pool of idle resources and ramped up upon request. In our experiment setup, we have abstracted away data consistency and rebalancing issues; however, state of the art solutions such as the previously mentioned Data Rebalance Controller (RDC) [12] could seamlessly be included into our modular architecture.

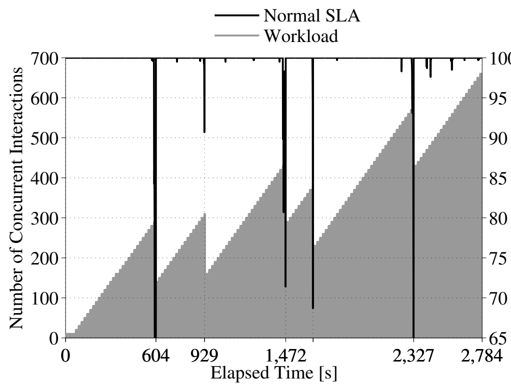
3.2 IAE Implementation Details

In order to provide maximal compatibility to the elastic application testbed, the IAE prototype was implemented based on a similar software stack as the testbed. Most of the prototype code was written in the Microsoft .NET environment while the mathematical logic of the WFM sub-model uses Matlab 2010. The multi-model controller code was deployed as a service on an IIS server, and the ODS was built in a Microsoft SQL Server backend. The learning module functionality in the ODS was designed as an append-only database

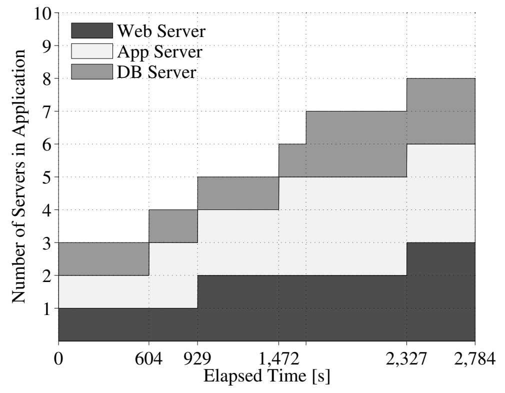
[1]We use a symbolic three-digit notation #Web/#App/#DB to denote the configuration; e.g., the smallest application size will be referenced as 1/1/1.



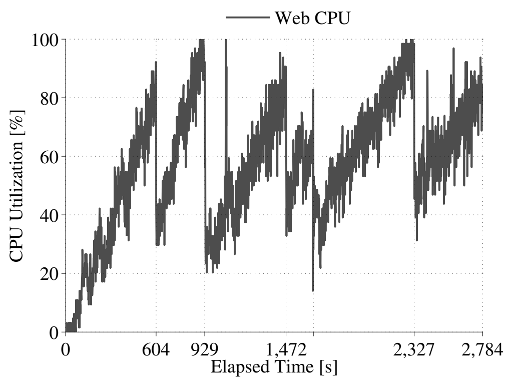
(a) Sensitive SLA-satisfaction (0.5s) vs. workload.



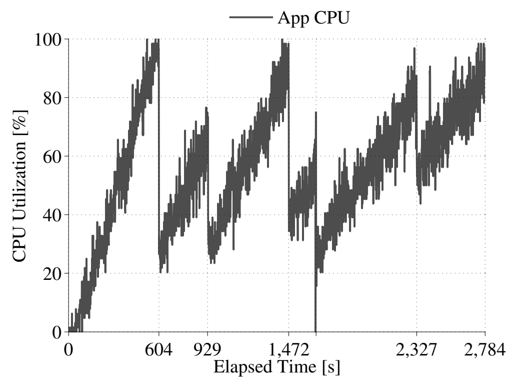
(b) Normal SLA-satisfaction (1s) vs. workload.



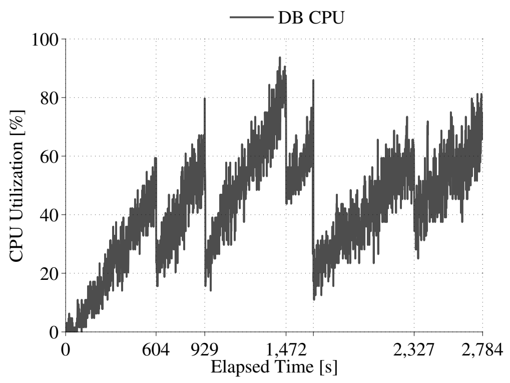
(c) Replication level of servers in application.



(d) Web server CPU utilization trace.



(e) Application server CPU utilization trace.

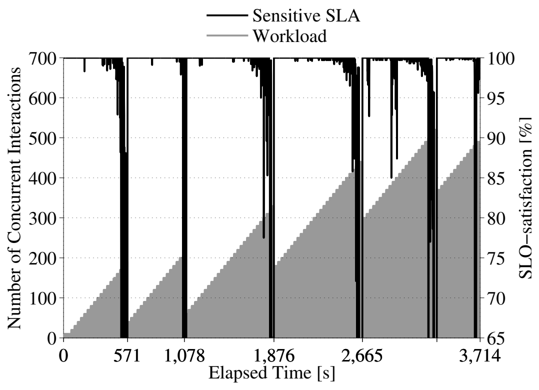


(f) Database server CPU utilization trace.

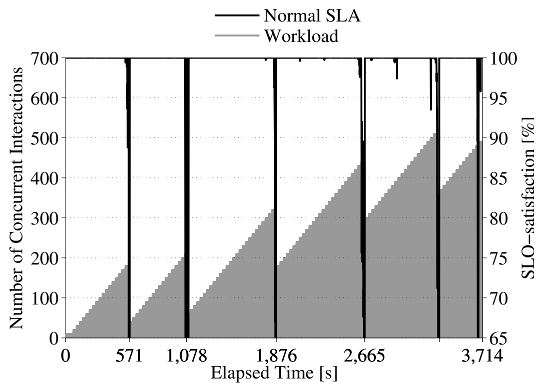
Figure 6: ODS initialization based on a synthetic workload with a uniform interaction distribution.

relation that captures application performance data in interaction-specific one-second throughput vectors. The scalar `@threshold` parameter was instantiated with 50 interactions per second. The EM query result set (see Section 2.1.2) is further processed by a second query that filters all configurations based on `support` of at least 30 distinct observations, `sla` of at least 90% on average, and by returning the cheapest configuration (i.e., addition of "**`TOP 1`**" to the query). Finally, the abstract `config` specification in the ODS relation was instantiated as three-integer notation that represents tier cardinality (i.e., the configuration), and the `cost` attribute was mapped as sum of all tier cardinalities.

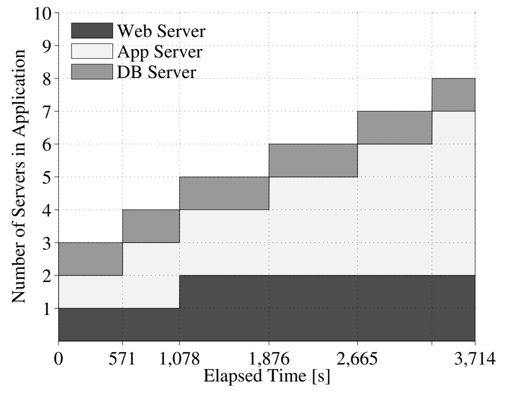
Due to the space constraints of this paper, we set the distance function weighting parameter $\alpha = 0$ and left the analysis of convergence properties of the EM as future work. The response time thresholds for the sensitive SLA and the normal SLA were instantiated to 0.5 and 1 second, respectively, based on the characteristics of the RUBBoS benchmark. SLA-satisfaction was calculated as percentage of requests that were processed within the response time threshold during a time interval of 30 seconds (intercepted from frontend logs). Similarly, the HSM averaging interval was set to 30 seconds.



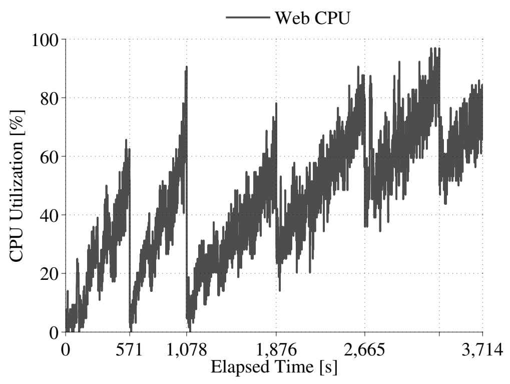
(a) Sensitive SLA-satisfaction (0.5s) vs. workload.



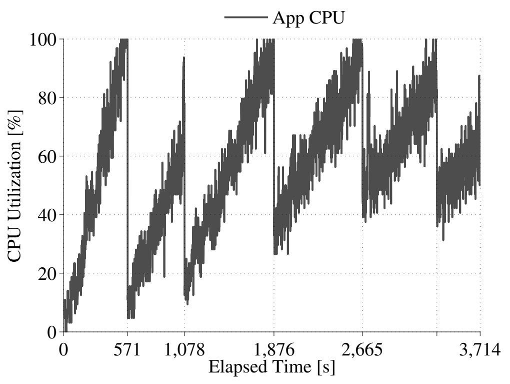
(b) Normal SLA-satisfaction (1s) vs. workload.



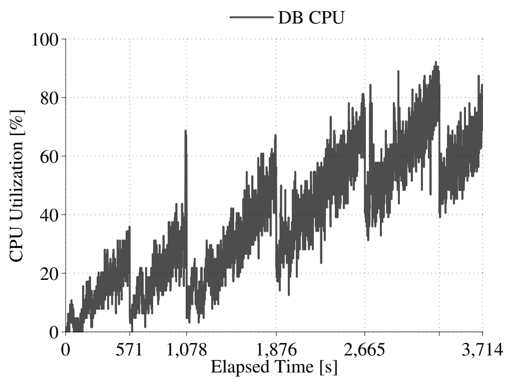
(c) Replication level of servers in application.



(d) Web server CPU utilization trace.



(e) Application server CPU utilization trace.



(f) Database server CPU utilization trace.

Figure 7: ODS initialization based on a synthetic workload with increased application server load.

3.3 Experimental Results

The following experimental results are divided into three parts. First, we introduce results that were generated with a synthetic (i.e., monotonically increasing step-function) workload in order to initialize the ODS in the IAE architecture. Second, we exemplify the characteristics of the EM scaling process based on the previously collected operational data. Third, we conclude the experimental evaluation by zooming out and exposing the IAE prototype to a one-week production system workload trace that illustrates the macro-characteristics of our approach.

3.3.1 Synthetic Workload Results

Here, we analyze results that were obtained with a monotonically increasing synthetic workload and an empty ODS. Consequently, the adaptation decisions are based on the input of the HSM. This approach serves two important purposes. First, we illustrate a state of the art control-based approach (i.e., HSC [12]) in our test environment. Second, we explore a practical strategy to fill the ODS with data and "initialize" our EM for the analysis that is shown in the following sections.

Both Figure 6 and Figure 7 are organized identically, and they depict the scale-up characteristic of the elastic n-tier application when exposed to an increasing interaction load. The difference between these two figures is that while the seven interactions are distributed uniformly in the former, the probability for Business Logic Requests (BLRs) was doubled in the latter. As discussed in the following, this change leads to a significantly higher load on the application tier, which ultimately causes different scaling characteristics. The workload traces are illustrated through the shaded area in Figures 6(a), 6(b), 7(a), and 7(b). The workload is generated according to a heuristic loop: Increment the number of concurrent interactions every 30 seconds by ten until the normal SLA-satisfaction in the last 30 seconds drops below 90 percent on average; then scale-up, drop the current workload level by 150 concurrent interactions, and restart the loop. Sensitive and normal SLA-satisfaction illustrate the sensor metric and the control-target, respectively. While Figures 6(c) and 7(c) show the number of VMs in each tier of the application during the scale up process, Figures 6(d)–6(f) and 7(d)–7(f) show the corresponding average CPU utilizations for each tier. These CPU utilizations are the sole decision metric for the HSM.

The analysis of Figure 6 confirms that the application load is balanced if all seven requests are distributed equally. Figure 6(c) shows a balanced scale-up pattern with nearly equal numbers of VMs per tier. As the workload scales from 10 to about 700 concurrent interactions, the elastic application scales from 1/1/1 to 3/3/2. However, the scale-up performance (i.e., maximal throughput per scaling step) is not monotonically increasing, which may seem surprising at first. In fact, the fourth configuration step (i.e., the addition of the third application server) results in a maximal achievable throughput level (380 ops/sec with 2/3/1) that is lower than the maximal throughput level in the previous scaling step (410 ops/sec with 2/2/1). The explanation for this phenomenon lays in the balanced performance among all tiers. Concretely, the increase in capacity of the application tier conversely increases the load in the database tier because the application servers process requests faster. This increased "pressure" on the database tier results in overall system saturation at an absolutely lower workload compared to the previous configuration. In fact, the CPU utilization analysis (Figures 6(d)–7(f)) reveal that the observed bottleneck cannot be explained with full CPU utilization and is most likely due to an I/O bottleneck between the two tiers. As illustrated in Figure 6(c), the overall system performance can be significantly increased (580 ops/sec with 2/3/2) once both tiers have been scaled up, and the communication is handled by a more balanced number of servers.

Since the probability of application tier intensive requests (i.e., BLR) has been significantly increased (from 14% to 25%) in the second scenario (Figure 7), the same synthetic workload level results in a higher number of VMs in the application tier (Figure 7(c)). As the workload scales from 10 to approximately 500 concurrent interactions, the elastic application scales from 1/1/1 to 2/5/1. The analysis of the application tier CPU utilization in Figure 7(e) corresponds to the CPU-bound implementation of the BLR. Consequently, the HSM operates well based on the average application tier CPU utilization, which results in an approximately monotonic throughput increase per scale-up step (Figures 7(b) and 7(a)). Solely in the last scaling step (from 2/4/1 to 2/5/1), the application performance is limited by an I/O bottleneck that corresponds to the previously

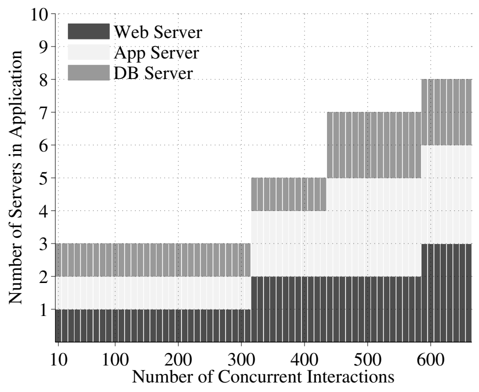


Figure 8: EM-based scaling with uniform interaction distribution (ODS initialized according to Figure 6).

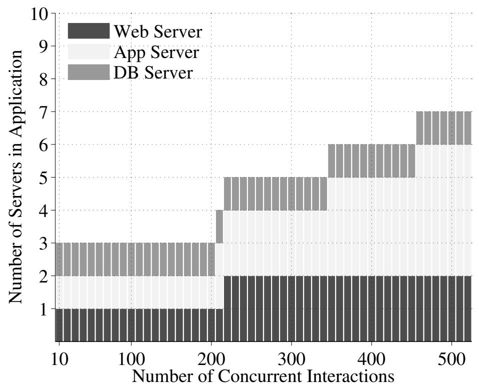


Figure 9: EM-based scaling with increased app. server load (ODS initialized according to Figure 7).

explained symmetric scaling phenomenon. Another important observation is the consistent relationship between sensitive and normal SLA-satisfaction. Naturally, the sensitive SLA-violations are a superset of the normal SLA-violations. This confirms the suitability of the sensitive SLA as a sensor metric for the pro-active control of the normal SLA, which is the true measure of QoS in our target environment.

3.3.2 Empirical Scaling Results

The results in this section illustrate the scaling decisions of the EM in contrast to the scaling decisions of the HSM in the previous section. Figure 8 shows that the EM is capable of making more complex scaling decisions than the HSM by scaling multiple tiers at the same time. For example, the EM automatically mitigates the aforementioned non-linear scaling characteristic and determines that it is more efficient to scale directly from 2/2/1 to 2/3/2 as the workload grows greater than 420 concurrent interactions. Another important difference between the EM and the HSM is shown in Figure 9. Despite the potentially available eighth VM, the EM does not scale beyond 2/4/1 because the previously observed 2/5/1 performance was not economical due to the I/O bottleneck.

The analysis of Figures 8 and 9 illustrates that the EM

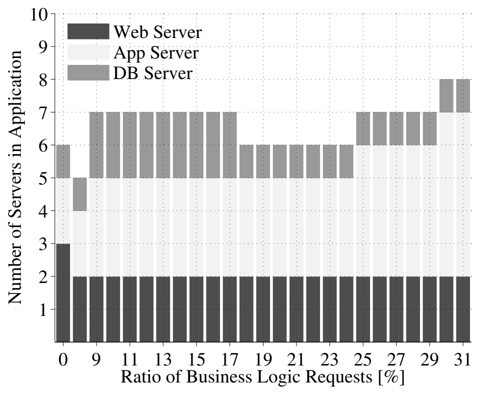


Figure 10: EM-based scaling with uniform workload and changing ratio of Business Logic Requests.

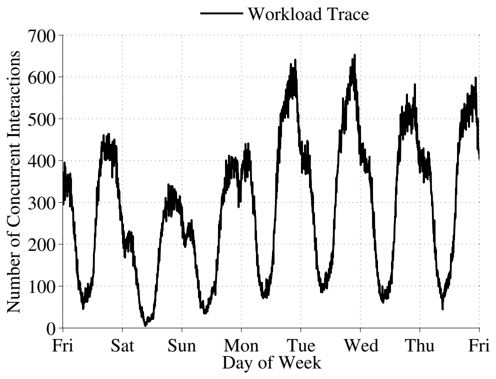


Figure 11: Assimilated `Wikipedia.com` workload.

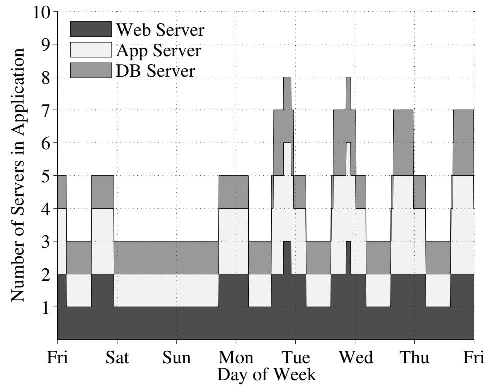


Figure 12: IAE application scaling based on the workload in Figure 11 and balanced interactions.

is not directly dependent on resource utilization metrics because it is able to directly map workload (x-axis) to economical configurations (y-axis). This property of the EM abstracts away the need to statically define an appropriate global resource utilization threshold such as 20% CPU utilization in Lim et al.'s paper [12]. Therefore, the EM approach enables operation at potentially higher resource utilization levels (see Figure 7(e)) as long as the SLA-satisfaction remains within desired boundaries (see Figure 7(b)).

In Figure 10 we analyze the EM with regard to a shifting workload mix. The x-axis shows the recorded ratio of BLRs and the y-axis shows the corresponding configuration at a workload greater or equal to 400. For this analysis the data in the ODS included all of the experiment runs that we have conducted to this point. From left to right, this figure illustrates the changing provisioning requirements as the bottleneck shifts from the DB tier to the application tier. Again, the EM is able to seamlessly adjust to these changing conditions, irrespective of whether the change happens gradually or periodically.

3.3.3 Real Workload Results

In this section, we analyze our adaptation engine prototype when exposed to a constant mix workload (balanced interactions), which is derived from absolute user numbers that were recorded for the `Wikipedia.com` website in May 2008 (Figure 11). For the purpose of our analysis, the real hourly trace was transformed to span 0–700 users, spline interpolation was used to generate 30 second granularity [8], and a Gaussian noise process was injected into the signal [6]. The analysis of Figure 12 confirms that the prototype is able to control the elastic application autonomously. It is noteworthy that low seasonal peaks such as on Sundays do not result in system scale-up beyond 1/1/1. Furthermore, most adaptation actions involve the addition of at least two VM resources in order to generate a balanced system cardinality, which corresponds to this particular workload mix scenario.

4. RELATED WORK

To the best of our knowledge, there are few papers on the integration of multiple automated control models. Urgaonkar et al.'s work [25] is an example, where they combine reactive control and queuing-based prediction. Despite the small number of previous papers (some discussed below), there is little doubt that combining multiple models to overcome their individual limitations can be beneficial. More generally, although we have not found automated control mechanisms for cloud management that use empirical models directly, our work on multi-model controller and ODS can be seen as an integration (with extensions) of three component technologies: control systems, queuing models, and automated learning. We briefly outline some relevant work from each area in this section.

Another paper that includes a multi-model controller uses an integral control technique called proportional thresholding for automated elastic storage in clouds [12]. Their scaling of the storage service is managed through a meta-controller combining a control system-based data rebalance controller and a horizontal scale controller. Their paper makes some simplifying assumptions, e.g., storage service QoS depends only on CPU utilization as the sole control metric. They also set the target CPU utilization of 20%, which may be reasonable for storage service, but is relatively low for cloud node utilization. Other control system-based approaches differ from our work because they assume the availability of continuously scaling resources in non-distributed environments [18, 19].

A well-established approach to performance prediction of computer systems is queuing models. Due to typical assumptions made (e.g., stationary arrival rates and workload-independent utilization ratios among system modules) to facilitate analytical solutions [9], classic queuing models have limitations with elastic workloads on n-tier applications [10, 21, 24]. Several research efforts have attempted to address these limitations (e.g., Mi et al.'s work on modeling burstiness in n-tier applications [16, 17], Urgaonkar et al.'s predictive and reactive provisioning, and Thereska et al.'s IRONModel [23]. As an example, the IRONModel is a robust queuing modeling architecture based on principles such as construction during system design and continuous reevaluation of accuracy during operation. Nonetheless, queuing theory requires a priori modeling of all performance relevant queuing stations and interaction types. This is not required by an empirical model such as ours, but our multi-model controller can integrate queuing models to handle the more stable workload periods.

Our empirical model includes an automated learning module that continuously inserts measured data into the ODS. This is different from machine learning-based approaches (e.g., Cohen et al. [4, 5] and Sahai et al. [20]), since we use the measurement data without necessarily finding a pattern or analytical representation. More recently, Armbrust et al. introduced SCADS [3] that uses machine learning for elastic storage system scaling. Our empirical model avoids the typical difficulties of machine learning with data noise-sensitivity and over-fitting since we use the measured data directly.

5. CONCLUSION

We described a multi-model controller for appropriately mapping virtual machine nodes to elastic n-tier applications in clouds. The main advantage of the controller is its support for an empirical model that provides assured adaptation decisions using measured performance data from previous runs of the application. An automated learning module continuously augments the empirical model knowledge base with measured data from new application runs. For configurations not yet covered by measurements, a simple heuristic model provides approximate solutions.

We implemented a prototype of the multi-model controller with the empirical model on a database platform (called ODS). The experimental evaluation of our prototype implementation suggests that the empirical model is able to adapt to changing performance determinants and to maintain constantly high SLA-satisfaction with guaranteed performance predictions based on measured data. As a concrete example, the empirical model demonstrated adaptation cost savings over the heuristic model by suggesting a multi-node adaptation action when multiple tiers approach saturation simultaneously. Our positive initial results suggest that an empirical model can be integrated with other adaptation algorithms (e.g., [10, 11]) using the multi-model controller, to achieve "the best adaptation of all models".

Although this paper focuses on horizontal scaling in IaaS clouds, our work is also applicable to other cloud service offerings. In the case of Software as a Service (SaaS), for example, our work can help alleviate the management problems faced by the cloud provider. In the case of vertical scaling scenarios, the configuration specification in the data store can be extended to include the number of VM instances, their type, and other descriptive information to support finer granularity adaptation.

6. REFERENCES

[1] Animoto's Facebook Scale-up. `http://blog.rightscale.com/2008/04/23/animoto-facebook-scale-up/`, 2008.

[2] M. Arlitt and T. Jin: A workload characterization study of the 1998 world cup web site. *Network '00.*

[3] M. Armbrust, A. Fox, D. A. Patterson, N. Lanham, B. Trushkowsky, et al.: SCADS: Scale-independent storage for social computing applications. *CIDR '09.*

[4] I. Cohen, M. Goldszmidt, et al.: Correlating instrumentation data to system states: a building block for automated diagnosis and control. *OSDI '04.*

[5] I. Cohen, S. Zhang, et al.: Capturing, indexing, clustering, and retrieving system history. *SOSP '05.*

[6] D. Feitelson: Workload modeling for computer systems performance evaluation. `http://www.cs.huji.ac.il/~feit/wlmod/`, 2011.

[7] M. Hedwig, S. Malkowski, and D. Neumann: Taming energy costs of large enterprise systems through adaptive provisioning. *ICIS '09.*

[8] M. Hedwig, S. Malkowski, et al.: Towards autonomic cost-aware allocation of cloud resources. *ICIS '10.*

[9] R. Jain: *The art of computer systems performance analysis.* John Wiley & Sons, Inc., 1991.

[10] G. Jung, K. Joshi, M. Hiltunen, et al.: Generating adaptation policies for multi-tier applications in consolidated server environments. *ICAC '08.*

[11] G. Jung, K. R. Joshi, M. A. Hiltunen, et al.: A cost-sensitive adaptation engine for server consolidation of multitier applications. *Middleware '09.*

[12] H. C. Lim, S. Babu, and J. S. Chase: Automated control for elastic storage. *ICAC '10.*

[13] S. Malkowski, M. Hedwig, D. Jayasinghe, C. Pu, and D. Neumann: CloudXplor: A tool for configuration planning in clouds based on empirical data. *SAC '10.*

[14] S. Malkowski, M. Hedwig, and C. Pu: Experimental evaluation of N-tier systems: Observation and analysis of multi-bottlenecks. *IISWC '09.*

[15] S. Malkowski, D. Jayasinghe, et al.: Empirical analysis of database server scalability using an n-tier benchmark with read-intensive workload. *SAC '10.*

[16] N. Mi, G. Casale, L. Cherkasova, and E. Smirni: Burstiness in multi-tier applications: symptoms, causes, and new models. *Middleware '08.*

[17] N. Mi, G. Casale, L. Cherkasova, and E. Smirni: Injecting realistic burstiness to a traditional client-server benchmark. *ICAC '09.*

[18] P. Padala, K.-Y. Hou, et al.: A. Merchant. Automated control of multiple virtualized resources. *EuroSys '09.*

[19] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, et al.: Adaptive control of virtualized resources in utility computing environments. *EuroSys '07.*

[20] A. Sahai, S. Singhal, and Y. B. Udupi: A classification-based approach to policy refinement. *IM '07.*

[21] C. Stewart, T. Kelly, et al.: Exploiting nonstationarity for performance prediction. *EuroSys '07.*

[22] RUBBoS: Bulletin board benchmark. `http://jmob.objectweb.org/rubbos.html`, 2008.

[23] E. Thereska and G. R. Ganger: IRONModel: robust performance models in the wild. *SIGMETRICS '08.*

[24] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, et al.: An analytical model for multi-tier internet services and its applications. *SIGMETRICS '05.*

[25] B. Urgaonkar, P. Shenoy, et al.: Dynamic provisioning of multi-tier internet applications. *ICAC '05.*

[26] P. Xiong, Y. Chi, S. Zhu, H. J. Moon, C. Pu, et al.: Intelligent management of virtualized resources for database systems in cloud environment. *ICDE '11.*

[27] P. Xiong, Z. Wang, G. Jung, and C. Pu: Study on performance management and application behavior in virtualized environment. *NOMS '10.*