EntomoModel: Understanding and Avoiding Performance Anomaly Manifestations

Christopher Stewart
The Ohio State University
cstewart@cse.ohio-state.edu

Kai Shen
University of Rochester
kshen@cs.rochester.edu

Arun Iyengar
IBM Watson Research Center
aruni@us.ibm.com

Jian Yin
Pacific Northwest National Labs
jian.yin@pnl.gov

***Abstract*—Subtle implementation errors or mis-configurations in complex Internet services may lead to performance degradations without causing failures. These undiscovered performance anomalies afflict many of today's systems, causing violations of service-level agreements (SLAs), unnecessary resource over-provisioning, or both. In this paper, we re-inserted realistic anomaly causes into a multi-tier Internet service architecture and studied their manifestations. We observed that each cause had certain workload and management parameters that were more likely to trigger manifestations, hinting that such parameters could be effective classifiers. This observation held even when anomaly causes manifested differently in combination than in isolation.**

Our study motivates *EntomoModel*, a framework for depicting performance anomaly manifestations. EntomoModel uses decision tree classification and a design-driven performance model to characterize the workload and management policy settings under which manifestations are likely. EntomoModel enables online system management that avoids anomaly manifestations by dynamically adjusting system management parameters. Our trace-driven evaluations show that manifestation avoidance based on EntomoModel, or *entomophobic management*, can reduce 98th-percentile SLA violations by 67% compared to an anomaly-oblivious adaptive approach. In a cloud computing scenario with elastic resource allocation, our approach uses less than half of the resources needed in static over-provisioning.

I. Introduction

The complexity of Internet services continues to rise, in terms of the variety of supported user workloads and the interactions between off-the-shelf software components. It is increasingly challenging to manage these services for high and predictable performance. Consider the following service-level agreement (SLA)—"*at least* 98% *of online transactions should be completed within* 2 seconds". A monthly performance report on 11 leading e-commerce services from July to December 2008 [10] suggests that only three of the services could have satisfied this SLA.

Recent research has made it easier to manage performance in the face of workload fluctuations [16], [22], [25], system evolutions [26], and hardware changes [15], [23]. However, situations where Internet services exhibit poor performance may also stem from mis-configurations, poorly implemented functions, or unexpected software interactions [2], [3]. These *performance anomaly manifestations* are rare but realistic, and it is widely accepted that many Internet services suffer from the manifestations of undiscovered performance anomalies.

In the first part of this work, we reinserted authentic anomaly root causes into a multi-tier Internet service architecture and studied their manifestations. In total, we examined 3,176 performance anomaly manifestations from 21,408 experiments with root causes reinserted in isolation and in combination. Our study found that for each root cause (or combination of causes) there were certain workloads and management policies that were more likely to trigger manifestations. This finding held even when the combination of multiple causes produced new, unexpected manifestations (emergent mis-behavior) and when the combination of multiple causes surprisingly masked each other's manifestations (emergent good behavior) [8], [14].

In the second part of this work, we propose *EntomoModel*, a framework for depicting the manifestations of undiscovered anomaly root causes in Internet services. EntomoModel uses decision-tree classification to depict anomaly manifestations over a multi-dimensional system parameter space. EntomoModel achieves high classification accuracy for realistic systems that are afflicted by multiple anomaly root causes. Further, our analysis shows that emergent behavior between performance anomaly causes has a positive effect on EntomoModel's accuracy.

EntomoModel enables a new system management approach in which anomaly manifestations are avoided by adjusting management policies in response to workload changes. A system augmented with such entomophobic management behaves like a person suffering from entomophobia—it avoids conditions where bugs are likely (hence, the name). Our experiments show that entomophobic management can integrate with adaptive control techniques [12], [15], [16] to substantially reduce SLA violations while using fewer resources. In summary, our contributions are:

1. We present an empirical study that highlights the characteristics of realistic performance anomaly causes and their manifestations in combination.
2. We present EntomoModel, a framework for identifying the operating conditions where anomalies manifest.

This work was supported in part by the U.S. National Science Foundation grants CCF-0448413, CNS-0615045, CCF-0937571, and by an IBM Faculty Award.

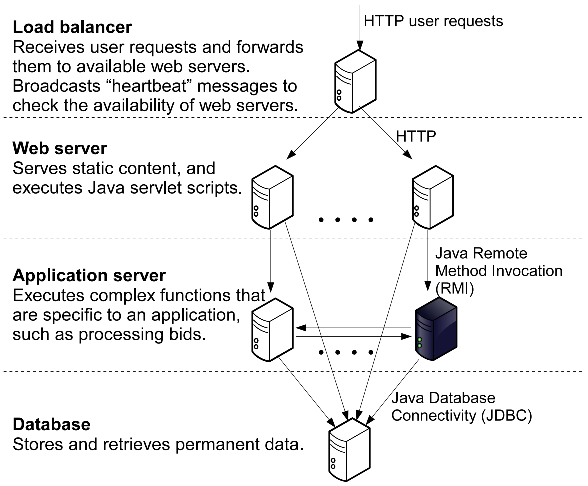

Fig. 1. Multi-tier architecture of our targeted services. Each machine runs the software component described to its left. Components at the web server and application server tiers may be replicated and run on multiple machines. Note that different application server machines may host different sets of functions (and thus different coloring in the figure). Arrows represent communication channels used during request execution. Arrow labels specify the communication protocols.

3. Finally, we describe an online management system that uses EntomoModel to steer system parameters away from settings that may cause anomaly manifestations.

II. A Study of Anomaly Manifestations

We reimplemented six realistic anomaly root causes from online bug repositories and studied their manifestations in isolation as well as in combination. Our goal was to identify characteristics that can help in anomaly-aware system management. In this section, we first describe our targeted Internet service architecture and our experimental setup. Then we discuss the results of experiments conducted with anomaly root causes in isolation and in combination across a wide range of operating conditions.

A. Target System

We targeted a typical multi-tier Internet service architecture shown in Figure 1. The software packages that we used for the four tiers were: the Distributor TCP load balancer (version 0.7), the Apache Tomcat web server and servlet container (version 5.5), the JBoss application server (version 5.0), and the MySQL database (version 4.1).

To extract general patterns in the anomaly manifestations under our multi-tier architecture, our study considered the impact of many confounding factors. One factor was the effect of different application services. Our empirical study explored manifestations across multiple application services, including TPC-W [5] and four versions of RUBiS [4]:

1. **TPC-W** is an e-commerce bookstore that supports requests ranging from browsing the best selling books to searching book titles to buying books. It is implemented using 15 Servlets that access the database through JDBC.
2. **RUBiS Servlets** is an implementation of the RUBiS online auction, a benchmark service that supports requests ranging from searching for items to viewing bids to buying items. It uses 26 Servlets.
3. **RUBiS CMP** implements RUBiS using J2EE Enterprise Java Beans (EJBs). EJBs execute complex application-specific functions, like processing bids. They may be selectively distributed on only certain application server nodes. The acronym *CMP* stands for container managed persistence; such EJBs cache only simple database queries. This version uses 22 Servlets and 27 EJBs.
4. **RUBiS BMP** also uses a mixture of EJBs and Servlets. The acronym *BMP* stands for bean managed persistence; these EJBs can cache complex database queries. This version uses 22 Servlets and 10 EJBs.
5. **RUBiS Session** uses EJBs that specify multiple database queries for the application server to cache—*i.e.*, the multiple queries used to process a user request. This version uses 22 Servlets and 17 EJBs.

We also considered the effect of four workload parameters that significantly affect the performance of Internet services. We hypothesized that these parameters would also affect anomaly manifestations.

- **INTENS:** The *request arrival intensity* is the request arrival rate divided by a reference capacity[1] of the targeted service. In our experiments, we allow five settings: low (approx. 10%), low-medium (30%), medium (50%), medium-high (70%), and high (90%).
- **REQMIX** controls the percentage of user requests that require more than one database access. We allow five settings: none (0%), light access (25%), medium access (50%), medium-heavy access (75%), heavy access (100%).
- **HTTP** controls the complexity of the HTTP protocol used. We allow three settings for this parameter: simple (HTTP 1.0), complex (HTTP 1.1), and very complex (HTTPS).
- **RMIPRO** controls the protocol used to invoke EJBs. In our experiments, we use two protocols: the JAVA RMI standard and a JBoss-specific protocol.

We also considered four system management parameters that could potentially mitigate manifestations.

- **REPLICA** controls the number of machines that the web server and application server are replicated on. The database and load balancer are not replicated. This

[1] We define the reference capacity of a service as the largest request arrival rate where at least 95% of the requests return successfully. We measure the reference capacity using a request mix that exercise all tiers in our services. The reference capacities for TPC-W, RUBiS Servlets, RUBiS CMP, RUBiS BMP, and RUBiS Session are 210, 240, 135, 145, and 135 requests per second, respectively.

parameter determines how many machines are used to run replicated web servers and EJBs.

- **REBOOT** is a boolean parameter that determines whether to restart the service before each test. Rebooting mitigates manifestations that arise gradually over the course of an experiment.
- **EJBGRP** controls the grouping of EJBs so that EJBs within each group are always co-located and replicated together. The EJBGRP parameter significantly impacts networked communications. Components placed in the same group enjoy fast-tracked RMI invocations without network transport of TCP processing.

  We support five possible EJB groups: all EJBs, CPU-heavy, CPU-lite, network-heavy, and network-lite. The first group (all) co-locates all EJBs together. The CPU-heavy/lite group co-locates EJBs that require the significant/insignificant processing per request. The network-heavy/lite group co-locates EJBs that exchange large/small database results. Each of our experiments involves two EJB groups (from the possible five). We allow flexible grouping in that the two groups are not necessarily exclusive (in which case some EJBs exist in both groups), nor do they need to cover all EJBs in combination (in which case the remaining EJBs are hosted in a designated application server).
- **EJBWGHT** controls the ratio used when EJB groups are assigned to application servers. For instance, a setting of 2:1 indicates that EJB group $\mathcal{A}$ is assigned to twice as many application servers as EJB group $\mathcal{B}$. By replicating certain groups more aggressively, we can mask anomalies that are caused by only a few machines. In total, we allow nine settings: 1:1, 2:1, 3:1, 4:1, 5:1, 1:2, 1:3, 1:4, 1:5.

Our experimental hardware platform is an 8-machine cluster in which each machine is equipped with two 1.266 GHz Intel Xeon processors, 2 GB memory, and 1 Gbps Ethernet. The load balancer and database run on their own isolated machines. Web servers and application servers can share machines. All databases fit within memory.

For a given setting of the workload, management, and application-service parameters (*i.e.*, a workload-policy setting), we compared the performance with an anomaly root cause enabled to the performance with the root cause disabled. We labeled a setting as an anomaly manifestation when the anomaly-enabled case exhibited a clearly lower performance (*i.e.*, the lowest measurement out of several tests for the anomaly-disabled case outperformed the best measurement for the anomaly-enabled case). Our performance metric was the ratio of successful requests to all incoming requests. Specifically, that is the percentage of requests that were completed within 2 seconds with an error-free response.

B. Root Causes in Isolation

We pulled six performance anomaly root causes, presented in Table I, from online bug repositories. Collectively, these causes span all layers in our multi-tier architecture. We used three selection criteria: First, we considered only performance anomalies. Second, each root cause had to be real, *i.e.*, it degraded performance in a real service. Third, each selected root cause had to have a well documented fix, *i.e.*, a code patch or configuration change that disabled the bug. The latter requirement allows a direct measurement on the effects of the root causes—a key step in our study.

For each root cause, we conducted experiments under 1,300 randomly selected workload-policy settings. Each experiment compared performance with the targeted root cause enabled and with it disabled, as described above. We observed that manifestations occurred in 5–11% of the tested settings. The average degradation caused by a manifestation ranged from 18–65%. These results confirmed our intuition that manifestations probably represent a small fraction of overall executions (*i.e.*, they occur infrequently), but their impact can be significant.

We also wanted to understand the correlation between system parameters and anomaly manifestations. We used the normalized information gain (or the uncertainty coefficient) [18] to measure correlation. Over a set of experiments S, the normalized information gain of a parameter p is:

$$NG(S,p) = 1 - \frac{\sum_{v \in values(p)} \frac{|S_v|}{|S|} \cdot H(S_v)}{H(S)}$$

where S_v, in our context, represented the experiments in S where parameter p was set to value v, and $H(S)$ was the Shannon information entropy of S. Our study concerned the binary categorization of observed performance as normal or manifestation, so the Shannon entropy was:

$$H(S) = -\alpha \cdot log_2(\alpha) - (1-\alpha) \cdot log_2(1-\alpha).$$

where α was the proportion of experiments in S where we observed manifestations. $1 - \alpha$ was the proportion of manifestation-free experiments (*i.e.*, normal).

Table I shows that, for each root cause, the most correlated system parameter explained at least 33% of manifestation uncertainty. However, the most correlated system parameter varied from one root cause to another. This result made sense because manifestations likely depend on the control flow specified by system parameters. For systems afflicted by multiple anomaly root causes, this result suggested that manifestations stemming from multiple root causes might have low correlation to system parameters. Low correlation could make it difficult to mitigate manifestations using system parameters.

C. Root Causes in Combination

We also compared performance without any anomaly root causes enabled to performance with several root causes enabled. An anomaly manifestation occurred when performance with the combined root causes enabled was clearly lower. We conducted these experiments under the same 1,300 workload-policy settings that we used to test the root causes in isolation.

Tag	Root cause description	Source Online Repository	Manifestation frequency	Performance degradation	Correlated parameter
LB	The implementation of round-robin load balancing distributed requests unevenly, starving the last web server listed in the configuration file. This bug manifested as slower response times for the HTTP requests sent to the other web servers.	Distributor version 0.6	11%	65%	REPLICA (43%)
LOC	Explicit attempts to access EJBs on remote machines were ignored if the application server on the local machine also hosted the EJB. This bug caused slower response times when local machines need to offload some of their workload. Note, our EJBs are stateless, so the redirection does not cause incorrect behavior.	JBoss.org JIRA #JBAS-1442	7%	48%	APP. SERVICE (34%)
RMI	Accesses to local EJBs unnecessarily incurred the same overhead as accesses to remote EJBs. This bug caused slower response times for services that use EJBs, especially services where a high percentage of EJB accesses are local.	JBoss.org JIRA #JBAS-1181	10%	32%	EJBGRP (66%)
TC	The connection timeout for some web servers was set too low (<200 ms) which caused the load balancer to occasionally mark them as unavailable. This bug caused 'connection timeout' errors for HTTP requests sent to the mis-configured server.	Tomcat archive #46D6F4A6	8%	18%	INTENS (35%)
THR	Each application server allowed only 10 concurrent EJB executions. Requests that issued multiple RMI incurred increased queuing delay. Sometimes requests could not proceed until other requests timed out (2-second delay).	JBoss.org JIRA #JBAS-4586	5%	44%	EJBGRP (72%)
DNS	The domain name server (DNS) did not map some cluster IP addresses to a domain name. Network communications with the unmapped machines were delayed whenever they passed through the DNS.	MYSQL Forums #11685	7%	21%	HTTP (60%)

TABLE I
THE ANOMALY ROOT CAUSES USED IN OUR STUDY. FOR EACH ROOT CAUSE, WE PROVIDE A DESCRIPTION, A REFERENCE, THE OBSERVED PERCENTAGE OF WORKLOAD-POLICY SETTINGS THAT EXHIBITED ANOMALY MANIFESTATION, THE AVERAGE OBSERVED PERFORMANCE DEGRADATION OF THE MANIFESTATIONS, AND THE WORKLOAD-POLICY PARAMETER MOST CORRELATED WITH MANIFESTATIONS WITH ITS INFORMATION GAIN. WE EXPERIMENTED WITH EACH ROOT CAUSE UNDER 1300 RANDOMLY-SELECTED WORKLOAD-POLICY SETTINGS.

We compared our results with root causes in isolation to our results with the root causes combined. Specifically, we classified each tested workload-policy setting across the following potential outcomes:

- **Normal-to-Normal** All experiments with the root causes in isolation exhibited normal performance and the experiment with the root causes combined exhibited normal performance.
- **Manifestation-to-Manifestation** One or more experiments with the root causes in isolation exhibited an anomaly manifestation and the experiment with the root causes combined exhibited an anomaly manifestation.
- **Normal-to-Manifestation** All experiments with the root causes in isolation exhibited normal performance but the experiment with the root causes combined exhibited a manifestation.
- **Manifestation-to-Normal** One or more experiments with the root causes in isolation exhibited a manifestation but the experiment with the root causes combined exhibited normal performance.

The latter two cases (normal-to-manifestation and manifestation-to-normal) reflect manifestations that cannot be explained from each root cause's standalone manifestations.

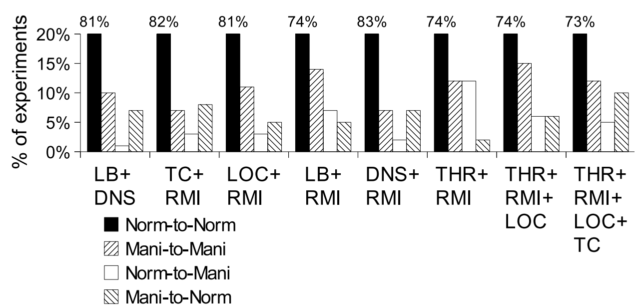

Fig. 2. Comparison of experiments with root causes in isolation to root causes in combination. 1,300 workload-policy settings were tested.

We call the normal-to-manifestation cases as *emergent mis-behaviors*. We then call the manifestation-to-normal cases *emergent "good" behaviors*. While emergent mis-behaviors of complex distributed systems have been studied in the past [14], little attention has been paid to the emergent manifestation-to-normal cases. These cases are particularly counter-intuitive and unexpected.

Figure 2 shows the distribution of tested workload-policy settings across our classification scheme. Surprisingly, we observed that manifestation-to-normal classifications occurred often, *e.g.*, in 8% of the tested settings in the TC+RMI combination. In five of the eight studied combinations, these emergent manifestation-to-normal classifications outnumbered normal-to-manifestation classifications. These emergent good behaviors contributed to an overall resilience to anomaly manifestations in our multi-tier architecture. Even with four root causes combined together, 83% of the tested workload settings were normal. Without emergent good behavior, the proportion of normal workload settings could have dropped to as low as 73%.

At first, we were surprised by the existence of emergent manifestation-to-normal conditions. How can performance-degrading anomalies combine to produce normal performance? Further, why would this happen more often than normal-to-manifestation conditions? To answer these questions, we investigated the reasons for emergent behavior in our study. While the exact reasons varied depending on which root causes were combined, we noted two properties of Internet services that helped to explain the phenomena. First, Internet services are measured according to a whole-system performance metric, *e.g.*, response time, even though their components (and anomaly root causes) operate on low-level resources, *e.g.*, CPU and network links. Emergent good behavior typically occurred when the low-level effects of an anomaly did not propagate to our whole-system performance metric. This matches established theory on emergent behavior [8]. The second property of Internet services that contributed to emergent good behavior is that they are designed to perform well under a variety of workload conditions. Emergent good behavior, like the examples given below, often occurred because the introduction of another anomaly changed low-level resource consumption in a way that allowed the system's robust performance features to mask anomaly manifestations.

Manifestation-to-Normal Example #1: The DNS root cause delayed communications between the load balancer and certain replicated web servers. When this root cause was reinserted in isolation, the communication delay manifested as response time degradation. However when we reinserted the DNS and LB causes in combination, we observed instances of manifestation-to-normal emergent behavior. Many of these instances occurred because LB unbalanced the workload for replicated web servers unaffected by the DNS cause, essentially trading CPU consumption on the unaffected nodes for communication delay. In many cases, the unaffected web servers could robustly handle the additional load.

Manifestation-to-Normal Example #2: When a heavily loaded web server ran on the same machine as an application server, the RMI root cause could increase the queuing time to access the CPU which increased response times. However, when TC forced timeouts on the heavily loaded web server, the high-performance design in the load balancer adapted by sending requests to lightly loaded web servers.

Root cause		**Anomaly classifications** Normal	Manifest
LB	actual normal	384	**12**
	actual manifest	**4**	45
LOC	actual normal	415	**0**
	actual manifest	**11**	19
RMI	actual normal	372	**22**
	actual manifest	**8**	37
TC	actual normal	366	**35**
	actual manifest	**22**	22
THR	actual normal	404	**8**
	actual manifest	**19**	14
DNS	actual normal	338	**74**
	actual manifest	**6**	27
LB+DNS	actual normal	393	**6**
	actual manifest	**7**	39
TC+RMI	actual normal	327	**76**
	actual manifest	**5**	37
LOC+RMI	actual normal	347	**30**
	actual manifest	**15**	53
LB+RMI	actual normal	318	**33**
	actual manifest	**13**	81
DNS+RMI	actual normal	304	**97**
	actual manifest	**6**	38
THR+RMI	actual normal	299	**39**
	actual manifest	**16**	91
THR+RMI+LOC	actual normal	319	**32**
	actual manifest	**22**	72
THR+RMI+LOC+TC	actual normal	340	**28**
	actual manifest	**15**	62

TABLE II
PREDICTION ACCURACY OF DECISION TREE CLASSIFICATION FOR COMBINED ANOMALY ROOT CAUSES. MANIFESTATIONS WERE IDENTIFIED USING THE APPROACH IN SECTION II. THE DECISION WAS CONSTRUCTED USING 855 WORKLOAD-POLICY SETTINGS AND EVALUATED ON 445 SETTINGS. RESULTS ARE SHOWN IN THE FORM OF A CONFUSION TABLE—THE UPPER-LEFT AND LOWER-RIGHT CORNERS REFLECT ACCURATE CLASSIFICATIONS WHILE THE UPPER-RIGHT AND LOWER-LEFT CORNERS REFLECT MIS-CLASSIFICATIONS.

III. Depicting Anomaly Manifestations

Anomaly manifestations can significantly degrade an Internet service's performance, potentially causing SLA violations. This motivates the need to predict and avoid anomaly manifestations before they happen. We present EntomoModel, a framework for depicting workload-policy settings under which manifestations are likely. Figure 3 illustrates our three-step approach as follows: 1) we collect a training set of workload-policy settings; 2) we label the training settings as "normal" or "manifestation" through offline experimentation; 3) we use the labeled training settings to construct a decision tree that classifies every workload-policy setting in the parameter space. The result is a comprehensive portrait of manifestations and normal settings over an entire parameter space for the targeted service.

This paper tackles two challenges for the EntomoModel framework. Section III-A describes our approach to classify a parameter space from training samples of workload-policy settings. In Section III-B, we address the challenge of labeling anomaly manifestations without being able to enable and disable anomaly root causes, *i.e.*, labeling manifestations from unknown causes.

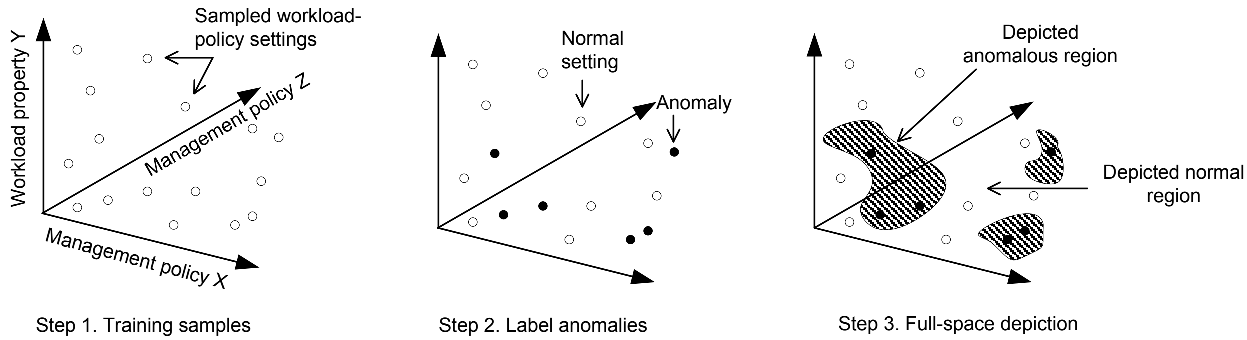

Fig. 3. EntomoModel's approach for performance anomaly depiction in a multi-dimensional space of workload-policy settings. Each parameter in the space is a workload property or system management policy.

A. *Decision Tree Anomaly Classification*

Decision trees classify points in an input feature space according to a set of discrete outcomes. For our purposes, the feature space is a set of workload and management parameters and a point in this space is a workload-policy setting. The two outcomes in our context are "normal" and "manifestation". Using the Iterative Dichotomiser 3 algorithm [17], our decision tree is generated in a top-down fashion by iteratively selecting the system parameter with highest information gain, partitioning samples based on their corresponding values of the parameter, and constructing a sub-tree for each partition until current samples all fall into the same category.

There are a number of other widely used classification techniques, such as naive Bayes classifiers, perceptrons, neural networks, Bayesian networks, support vector machines, and hidden Markov models. Our decision to use decision trees as the basis for EntomoModel was reached from qualitative (not quantitative) considerations. Some specific factors were: 1) decision trees did not require any a-priori assumptions about the anomaly manifestations patterns in our systems, 2) decision trees have been successfully tested on noisy data in prior systems studies [11], [26], and 3) after they have been built, decision trees can be evaluated quickly to provide hints during system management. Our choice of decision trees does not mean that it is the only appropriate technique for classifying bug manifestations in EntomoModel. The exploration of the different machine-learning algorithms for performance bug classification is an interesting area for future work.

1) Experimental Validation: To evaluate the prediction accuracy of EntomoModel's decision tree classification, we leveraged our collection of realistic anomaly root causes. Like in Section II, we labeled a workload-policy setting as a performance anomaly manifestation if the performance with a targeted root cause enabled was lower than the performance with the root cause disabled. Using this method, we classified 1,300 workload-policy settings. We used 885 (about two thirds) of the classified settings to train the classifier and 445 to test its accuracy.

The top portion of Table II shows classification results when the anomaly root causes were reinserted in isolation. The decision tree correctly classified most settings, including rare anomaly manifestations. For example, the mis-classification rate for LOC in isolation was only 2.5%. These results were in line with previous studies that successfully used decision trees to classify system evolutions [26] and application-level failures [11].

The EntomoModel decision tree classifier was also accurate when root causes were enabled in combination (the bottom portion of Table II). The median ratio of true manifestations (lower right) to false normals (lower left) was 5.5. This ratio ranged from 3.2–6.3 across all of our experiments. These results speak to the agility of decision tree classifiers. Even though manifestations from different root causes are correlated with different parameters, as shown in Section II, our classification approach still identified anomalous regions of the workload-policy parameter space.

2) Impact of Emergent Behaviors: We studied the impact of emergent good behaviors (manifestation-to-normal settings described in Section II-C) on decision tree classification. Specifically, we compared decision tree results when the bugs were combined to the results of a hypothetical scenario where emergent good behaviors were labeled as manifestations. Figure 4 shows that emergent behavior decreased mis-classification rates by up to 50%.

To gain intuition for the results in Figure 4, we examined the conditions that exhibited emergent good behaviors when we combined the LB and RMI root causes. That is, we examined only the conditions classified as manifestation-to-normal for this combination of root causes. We found that these conditions were evenly distributed across all system parameter settings. No parameter offered information gain greater than 1%. Our interpretation of Figure 4 is that emergent good behaviors essentially removed manifestations without bias, making it easier to classify the resulting fewer manifestations.

B. *Depiction of Unknown Anomaly Causes*

In the real world, anomaly causes are typically unknown and fixes are certainly unavailable. We extended EntomoModel with an anomaly identification heuristic that classifies a workload-policy setting as a potential manifestation if it exhibits poor performance against expectation. Specifically, we screen out workload-policy settings that perform poorly due to inherent capacity overloading using a whole-system

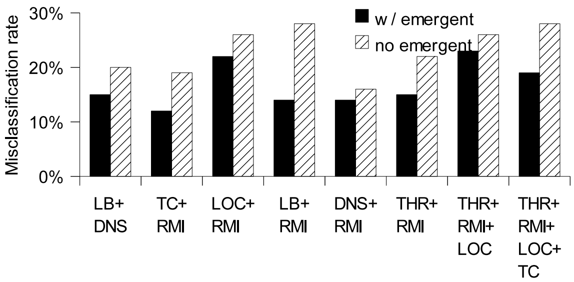

Fig. 4. The effect of emergent behavior on the misclassification rate for anomaly manifestations. Lower misclassification rate is better.

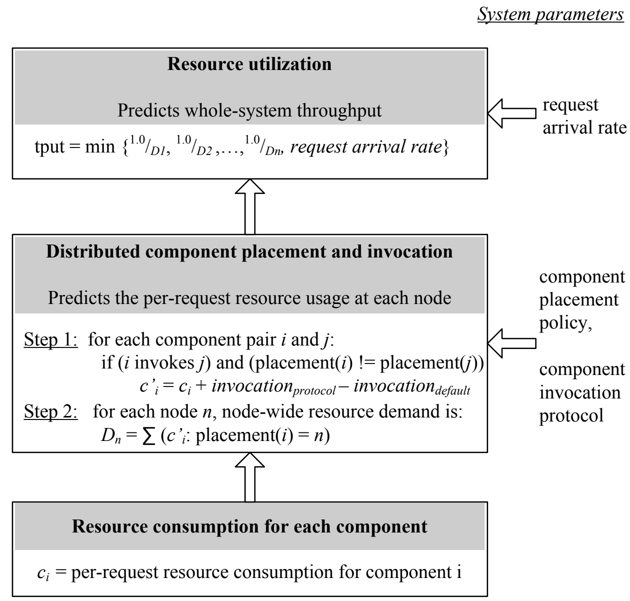

Fig. 5. A stacked performance expectation model for multi-component distributed online services. Each level transforms lower-level system metrics into its prediction targets based on the impact of input system parameters (on the right).

performance expectation model based on the high-level design of multi-tier Internet services [25]. Under our heuristic, a workload-policy setting that exhibits low performance that actually matches expectations is not due to performance bug manifestation (*i.e.*, we consider it "normal").

Performance expectations can be constructed using past efforts in online service performance modeling [13], [25] as well as simple operational analysis [7]. In this paper, we utilize a performance expectation model for multi-tier Internet services that considers system management parameters including distributed component placement, remote invocation, and component replication level [25]. It is sufficient to support our system case study in this paper. However, emerging systems with radically different architectures [19], [24] may require new design-based performance models before our framework can be applied.

1) Performance Expectation Model: Our performance model accepts the specification of an input workload and management policy setting, and then outputs a prediction of application-level throughput. To reduce the modeling complexity, we employ a stacked performance model (shown in Figure 5) in which low-level system metrics are assembled into high-level metrics according to multiple largely independent measurements or sub-models. Below we describe each part of our model. We also provide a running example of performance expectations at each step.

- Resource Consumption Measurement—
The first step in our model is to measure the resource consumptions of each application component. Specifically, we would like to capture the innate resource needs that are independent of the system management policies. To start our running example, consider an application with two components. Measurements from this step may yield CPU observations of 18% and 10% for each component under the default configuration with base workload intensity of one request per second.
- Component Invocation Overhead—
Component invocations have a complex relationship with the component placement. Specifically, application servers often implement fast-path invocations that consume few resources when interacting components are co-located on the same physical machine. For simplicity, we assume such local invocation cost is zero. We measure the costs of remote invocation for two J2EE protocols as the difference between the summed resource consumption of two interacting components first run separately and then co-located.
- Distributed Component Placement—
Adjusting the component placement policy allows us to predict the resource consumption of each distributed node. Specifically, we sum the expected per-component resource consumption for each component placed on a particular node. Assume the components in our running example are placed on separate nodes and the component with heavier utilization invokes the other. 2% overhead may be added to the invoking component, leaving us with 20% and 10% resource utilization on the two nodes respectively.
- Resource Utilization Model—
Our throughput model is based on the Utilization Law [7] which describes system throughput as the quotient of resource utilization divided by the average per-request resource demand. The saturation throughput for a particular node occurs at 100% resource utilization. From the previous level of our stacked model, we have derived an estimate of the average per-request demand ($\mathcal{D}_{n,r}$) on a particular type of resource r (*e.g.*, CPU and network bandwidth) at each node n in the system. In the simple case without adaptive load balancing, the maximum throughput is reached as soon as one of the servers cannot handle any more load. Therefore, our model estimates the maximum throughput as the lowest saturation rate for all

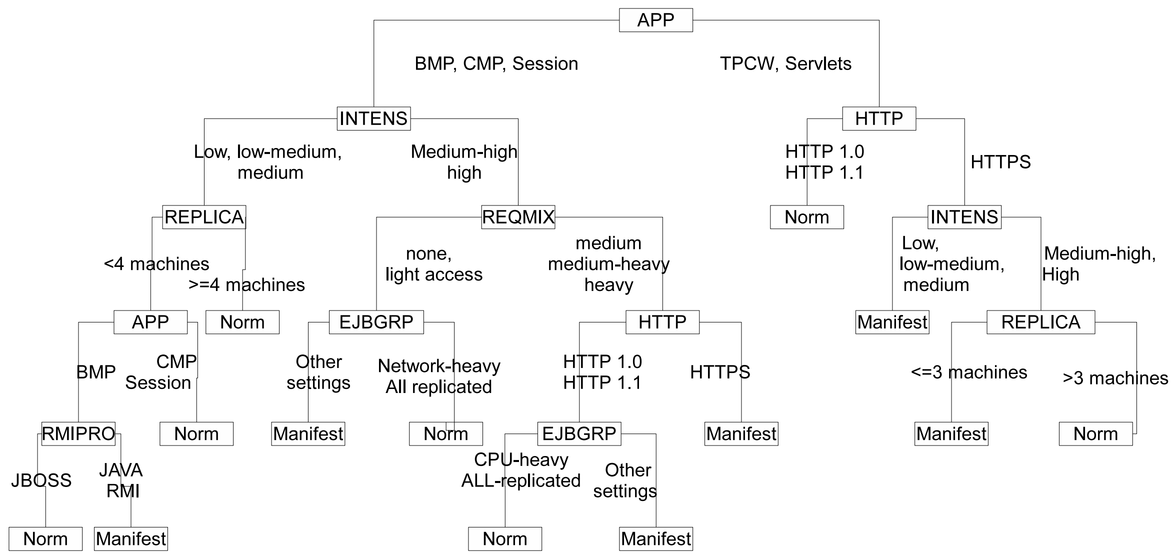

Fig. 6. The EntomoModel decision tree depiction for the system parameter space described in Section II. The target system is free of any of the known performance bugs.

resource types at all servers:

$$\tau_{\max} = \min_{\text{for all server nodes } n, \text{ resource types } r} \frac{100\%}{\mathcal{D}_{n,r}} \quad (1)$$

Given a request arrival rate τ_{workload}, the expected throughput should be the smaller of $\tau_{\max}$ and τ_{workload}. Consider our running example of a two-node system in which the average per-request CPU utilization was 20% and 10% respectively. For any workload intensity below 5 requests/second, we would expect all incoming requests to be serviced. Under heavier workloads, throughput would be expected to be around 5 requests/second.

2) Experimental Results and Validation: We show EntomoModel's depictions of anomaly manifestations due to unknown anomaly causes. The target system is free of any of the known performance bugs listed in Table I. From a set of training workload-policy settings, we label their normal or anomalous states by comparing measured performance and the performance model expectation. Figure 6 shows the EntomoModel decision tree depiction for the whole system parameter space. Each path from the root (split on the APP parameter) to a leaf represents the prediction of a class of workload-policy settings as potential manifestations or likely normal. We observe similar properties as in our empirical study with real performance bugs. First, six of the eight workload and management parameters are correlated with at least one potential manifestation. Second, the manifestations occur across a range of workload-policy settings: including under well-provisioned cases (*e.g.*, REPLICA $>$ 3) and light request arrival intensities (*e.g.*, INTENS = low).

Tested on 445 workload-policy settings (not used in training), EntomoModel's depiction of manifestations due to unknown anomaly causes showed low misclassification. Specifically, we observed that the misclassification rate was only 9%. The ratio of true anomaly classifications to false normal classifications was 4.5. These results are in line with the findings when the root cause was known, which suggests that our model captures the intended performance of multi-tier architectures.

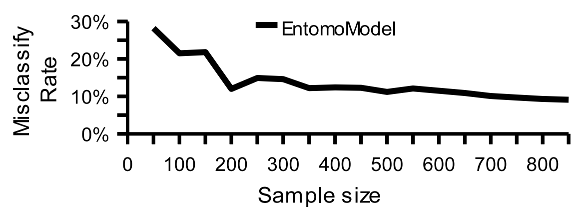

Fig. 7. The misclassification rate of EntomoModel using different sample sizes. Sample size refers to the numbers of system measurements used to train the EntomoModel's decision tree classifier.

3) Impact of Sample Size and Parameter Selection: Our results thus far have used measurements under 885 randomly sampled workload-policy settings (approximately $\frac{2}{3}$ of 1,300 sampled settings) to train the decision tree classifier that underlies EntomoModel. We used the remaining 445 samples to evaluate the classifier's accuracy. Figure 7 shows the accuracy of our decision tree classifier on the same 445 samples but with smaller sample sizes. EntomoModel converges to a misclassification rate below 12% when the training sample exceeds 500 settings.

We also studied the effect of using fewer system parameters during decision tree classification. First, we prevented the ID3 algorithm from splitting on the management parameters described in Section II-A and then we prevented the algorithm from splitting on the workload parameters. Using 885 samples, the workload-only tree achieved a misclassification rate of 21% and the management-only tree achieved a misclassification rate of 28%. While Figure 6 showed that the combination of management and workload parameters could produce an accurate classifier, these results showed that considering a wide range of parameters was essential for EntomoModel's accuracy.

IV. Avoiding Anomaly Manifestations

EntomoModel predicts potential anomaly manifestations (anomalies) across a range of workloads and management policies. This enables a new approach to system management in which EntomoModel is queried to avoid anomalies during online execution. Here, EntomoModel works in conjunction with existing adaptive or model-driven management approaches that address other factors that affect performance [16], [25]. *Entomophobic management* is beneficial because anomaly manifestations may cause violations of service-level agreements (SLAs), and existing management approaches may not properly address such manifestation-induced SLA violations.

We used the multi-tier architecture described in Section II to implement an adaptive management framework for the RUBiS BMP auction service. We subjected the service to a sequence of realistic workload settings derived from a publicly available trace [22], [23]. We allowed the REPLICA and EJBGRP parameters to be changed in response to the changes in the workload. The REPLICA parameter, which controls the number of server machines used to service the workload, could take any value from 1–6. The EJBGRP policy could be ALL replicated, CPU-heavy co-located, or network-heavy colocated. Our goal was to adjust these parameters to minimize SLA violations. Practical SLAs may be specified in different ways. In our study, an SLA specifies the minimum percentage of requests that must respond correctly within a response-time bound of 2 seconds.

We incorporated EntomoModel into the adaptive management approach (called entomophobic management) as illustrated in Figure 8. Our approach works on fixed-length time intervals (*e.g.*, 3 minutes). Performance observation over each interval is used to derive adaptive management policy. System re-configurations, if desired, are made at interval boundaries. Specifically, our adaptive management approach follows one of three paths:

1. If the percentage of successful requests is 100% for three consecutive intervals, our adaptive management reduces the REPLICA parameter by 1 (to avoid over-provisioning resources).
2. If there was no SLA violation, our adaptive management makes no changes to the EJBGRP or REPLICA parameters.
3. If there was an SLA violation, our adaptive management queries EntomoModel to determine if the cause was due to an anomaly manifestation. If so, we use EntomoModel to identify an alternative setting of the EJBGRP or REPLICA parameters. If EntomoModel determines that the SLA violation was not caused by a manifestation, we assume the problem is related to under-provisioned resources and increase the REPLICA parameter by 1.

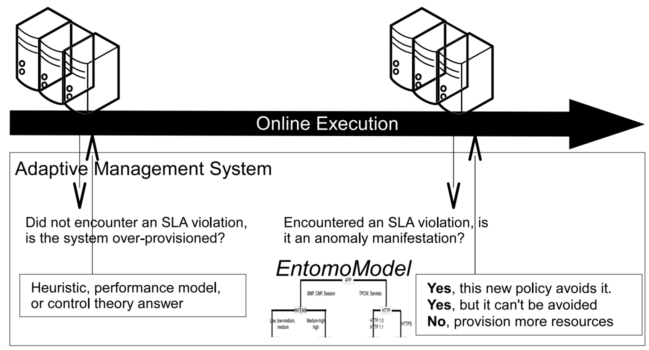

Fig. 8. An overview of EntomoModel used in adaptive management. EntomoModel is used to determine if a triggering SLA violation was caused by a potential anomaly manifestation. Management policy can be changed to avoid a repeat violation. The EJBGRP and REPLICA parameters can be changed only after a test completes, not during a test. Other parameters in our multi-tier architecture described in Section II are fixed. Specifically, we run the RUBiS BMP auction service with the load balancer configured to accept HTTP 1.1 requests. Application servers are configured to use the JBoss-specific RMI protocol and to distribute server resources to EJBs evenly. The target system is free of any of the known performance bugs. We use the EntomoModel anomaly manifestation predictions illustrated in Figure 6.

We compare our adaptive EntomoModel-based management to three alternative approaches.

- The *anomaly-oblivious adaptive* approach responds to all SLA violations by increasing the REPLICA parameter. The EJBGRP parameter is fixed to the CPU-Heavy/ALL-Replicated setting which offers the highest reference capacity. This approach assumes that performance problems can always be solved by using more machines.
- The *over-provisioning* approach takes the "add more machines" philosophy to the extreme. Here, the REPLICA parameter is always set to 6. The system is not dynamically adjusted at all.
- The *partial entomophobic* approach uses EntomoModel to determine if a violation is caused by a manifestation-induced anomaly. However, this approach is only allowed to adjust the REPLICA parameter.

A. Experimental Setup

We subjected the services to a sequence of realistic workload settings derived from a publicly available trace [22], [23]. The trace provides a non-stationary sequence of integers that correspond to the sequence of request types seen by a real Internet service. The trace website provides instructions on mapping these integers to RUBiS request types according to the popularity of request types in original service and in RUBiS. In total, we produced a sequence of 10,000 settings

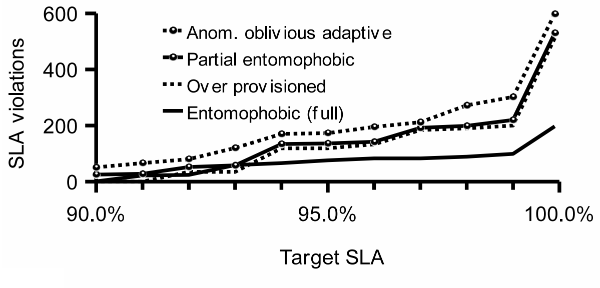

Fig. 9. The number of repeat SLA violations encountered using different management approaches. Each point reflects the total number of repeat SLA violations for our RUBiS application subjected to 10,000 trace-driven request rates and mixes. Using entomophobic management, our RUBiS application avoids repeat SLA violations in 99.1% of trace-driven workload settings.

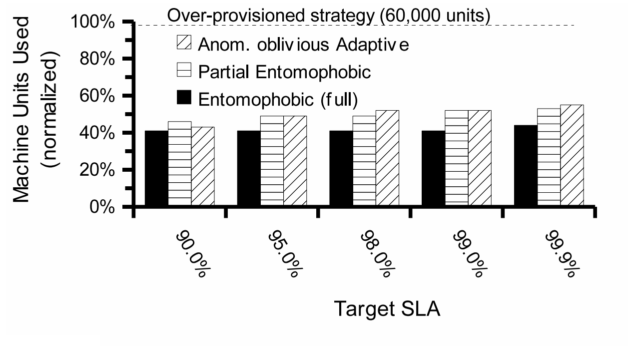

Fig. 10. The number of Machine Units used using different management approaches. The high watermark is when the percentage of successful requests is 100% for more than 3 consecutive intervals. A machine unit indicates the use of a server machine for 1 3-minute interval. The default over-provisioning approach uses 60,000 machine units (indicated by the dotted line). Using a pricing structure similar to Amazon EC2 (at $0.1 per unit) the full entomophobic management would save $3543 compared to the over-provisioned approach (while encountering fewer SLA violations).

of the REQMIX parameter (described in Section II). The sequence changes gradually over time. 70% of the adjacent settings in the produced REQMIX trace do not change or differ only slightly.

Also, we varied the request arrival rate in a sinusoidal fashion according to real-world diurnal fluctuations. More concretely, our request arrival rate makes about 200 fluctuations between 10 requests per second and 130 requests per second, with a step size of 10 requests per second. Our workload generator issued HTTP 1.1 requests according to the specified request mix and request rate settings from the trace. Tests at each setting were applied for 3 minutes. We simplify our experiments by stopping operations while the system is reconfigured.

B. Results

Figure 9 demonstrates the benefit of entomophobic management for avoiding SLA violations. Compared to the anomaly-oblivious adaptive approach, full entomophobic adaptive management reduces SLA violations by 62–67%. Compared to the over-provisioned approach, full entomophobic adaptive management reduces SLA violations by up to 62% when the SLA requirement on the percentage of successful requests is set to 99.9%. Full entomophobic adaptive management made 758 changes to the EJBGRP parameter and 251 changes to the REPLICA parameter in order to avoid potential SLA violations. Results with the partial entomophobic approach suggest that by using EntomoModel to identify potential performance manifestations, an adaptive approach that only changes the REPLICA parameter could get close to the performance of over-provisioned management.

An implicit sub-goal of our adaptive approach is to efficiently set the REPLICA parameter to avoid wasting resources. In practice, over-provisioning resources can be costly. We define the metric of a *machine unit* to indicate the use of 1 server machine for 1 test. A machine unit is intended to be an elastic resource allocation unit on a cloud platform, such as EC2 [1]. Figure 10 compares the number of machine units used by entomophobic management and the anomaly-oblivious adaptive approach. Under stringent SLA requirements of 99% and 99.9%, the full entomophobic management approach uses 22% and 19% fewer machines. This is because it is able to change the EJBGRP parameter to avoid potential anomaly manifestations without increasing the cluster size. Results with partial entomophobic management suggest that an adaptive anomaly-aware technique that changes only the REPLICA parameter can offer substantial improvements in resource usage compared to the over-provisioned approach.

V. Related Work

EntomoModel is essentially a predictive system model that improves automatic performance management. Generally, this topic has received a lot of attention in the research community and industry [9]. Menasce and Almeida have codified today's best practices with respect to queuing theory models and non-elastic cost models [13]. Stewart *et al.* [22], [23], [25] and Urgaonkar *et al.* [27] described the performance effects of varying important factors in multi-tier architectures, like processor cache size, request mix, and distributed component placement. Cohen *et al.* [6] use machine learning techniques to automatically predict SLA violations. These models are often integrated with control theory [12], [15], [16], [21] to manage the online performance of multi-tier services. EntomoModel builds upon these contributions by considering the effect of known and unknown performance anomalies. We showed that EntomoModel, in combination with adaptive policy, can significantly reduce SLA violations.

Researchers have also attempted to reduce the manual effort required to find and remove the root causes of performance anomalies. Shen *et al.* [20] attributed anomalous performance variation across system configurations to low-level system metrics which were useful for root cause analysis. Thereska *et al.* [26] used a hybrid machine-learned and human-designed performance model, based on decision trees, to find root causes in a storage file system. Kiciman and Fox [11] used decision trees to find availability degrading anomaly root causes in early multi-tier systems. Our study provides new insight on the runtime behavior of anomaly root causes, which can improve the heuristics of these measurement-based debugging systems.

Mogul examined emergent mis-behavior in today's computer systems via examples in networking, distributed systems, and operating systems [14]. The work categorized the causes and manifestations of emergent mis-behaviors, like those observed in our study. We build upon this work by studying both emergent mis-behavior *and* emergent good behavior across a range of operating conditions. Our study provides new empirical evidence on the existence and frequency of emergent behaviors in multi-tier architectures.

Some companies provide detailed reports when their customers experience severe performance anomalies [2], [3]. These reports share our goal of providing empirical data on real-world anomalies. Sometimes, they describe root causes similar to the anomalies studied in this paper, albeit at a much larger scale. For example, a performance anomaly that temporarily brought down Amazon S3 [2] was caused by a complex interaction between two root causes: 1) the system configuration placed critical account verification components on the same machines as CPU-intensive authentication components and 2) monitoring software overlooked a spike in authentication requests. These causes would have been benign in isolation, but in combination, a severe performance anomaly emerged. Our study of anomaly manifestations across a range of operating conditions complements and confirms these experience reports.

VI. Conclusion

This paper presents *EntomoModel*, a framework that systematically depicts the workloads and management policies under which potential performance anomalies are likely to manifest. EntomoModel uses a design-driven model of Internet services to label potential manifestations without knowing the underlying root cause. These labels are then fed into a decision tree classifier that depicts potential manifestations across a whole system parameter space. Using known anomalies as ground truth, we showed that EntomoModel's classifications have a low misclassification rate. Further, EntomoModel is accurate in realistic settings when multiple anomaly root causes are active, in part because of the effect of emergent behavior between anomaly root causes. EntomoModel is useful during online system management to avoid anomaly manifestations that could cause SLA violations. In our experiments, such entomophobic management reduced manifestation-caused SLA violations by 58–67% compared to alternative approaches.

References

[1] Amazon elastic compute cloud. http://aws.amazon.com/ec2/.
[2] Amazon web services developer community. http://developer.amazonwebservices.com/connect/thread.jspa?threadID=197%14&start=75&tstart=0.
[3] More on today's gmail issue. http://gmailblog.blogspot.com/2009/09/more-on-todays-gmail-issue.html.
[4] Rice university bidding system. http://rubis.objectweb.org/.
[5] Transaction processing performance council: E-commerce benchmark. http://www.tpc.org/tpcw/.
[6] I. Cohen, M. Goldszmidt, T. Kelly, J. Symons, and J. Chase. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *6th USENIX Symp. on Operating Systems Design and Implementation*, San Francisco, CA, Dec. 2004.
[7] P. Denning and J. Buzen. The operational analysis of queueing network models. *ACM Computing Surveys*, 10(3):225–261, Sept. 1978.
[8] J. Goldstein. Emergence as a construct: History and issues. *Emergence*, 1:49–72, Apr. 1999.
[9] IBM autonomic computing. http://www.research.ibm.com/autonomic/.
[10] Keynote Systems. Transaction performance index for online retail web sites electronics. http://keynote.com/keynote_competitive_research/performance_indices/ret%ail/retail_electronics_index.html, 2008.
[11] E. Kiciman and A. Fox. Detecting application-level failures in component-based Internet services. *IEEE Trans. on Neural Networks*, 16, Sept. 2005.
[12] X. Liu, J. Heo, L. Sha, and X. Zhu. Adaptive control of multi-tiered web application using queueing predictor. In *Network Operations and Management Symposium*, 2006.
[13] D. Menasce and V. Almeida. *Capacity Planning for Web Services: Metrics, Models, and Methods*. Prentice Hall, 2001.
[14] J. Mogul. Emergent (mis)behavior vs. complex software systems. In *First European Conf. on Computer Systems*, Leuven, Belgium, Apr. 2006.
[15] P. Padala, K.-Y. Hou, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, and A. Merchant. Automated control of multiple virtualized resources. In *4th European Conf. on Computer Systems*, pages 13–26, Nuremburg, Germany, Mar. 2009.
[16] P. Padala, K. G. Shin, X. Zhu, M. Uysal, Z. Wang, S. Singhal, A. Merchant, and K. Salem. Adaptive control of virtualized resources in utility computing environments. In *Second European Conf. on Computer Systems*, pages 289–302, Lisbon, Portugal, Mar. 2007.
[17] J. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.
[18] C. Sarndal. A comparative study of association measures. *Psychometrika*, 39(2):165–187, June 1974.
[19] K. Sauer, A. Ganapathi, C. Reiss, A. Constantin, A. Fox, M. Jordan, and D. Patterson. Automatic workload evaluation (awe): Predicting web 2.0 workload behavior. In *Poster Session of SOSP*, Big Sky, Montana, Oct. 2009.
[20] K. Shen, C. Stewart, C. Li, and X. Li. Reference-driven performance anomaly identification. In *ACM SIGMETRICS Int'l Conf. on Measurement and Modeling of Computer Systems*, Seattle, WA, June 2009.
[21] K. Shen, H. Tang, T. Yang, and L. Chu. Integrated resource management for cluster-based Internet services. In *5th USENIX Symp. on Operating Systems Design and Implementation*, Boston, MA, Dec. 2002.
[22] C. Stewart, T. Kelly, and A. Zhang. Exploiting nonstationarity for performance prediction. In *Second European Conf. on Computer Systems*, Lisbon, Portugal, Mar. 2007.
[23] C. Stewart, T. Kelly, A. Zhang, and K. Shen. A dollar from 15 cents: Cross-platform management for Internet services. In *USENIX Annual Technical Conf.*, Boston, MA, June 2008.
[24] C. Stewart, M. Leventi, and K. Shen. Empirical examination of a collaborative web application. In *IEEE International Symposium on Workload Characterization*, Seattle, WA, Sept. 2008.
[25] C. Stewart and K. Shen. Performance modeling and system management for multi-component online services. In *Second USENIX Symp. on Networked Systems Design and Implementation*, Boston, MA, May 2005.
[26] E. Thereska and G. R. Ganger. IRONModel: Robust performance models in the wild. In *ACM SIGMETRICS Int'l Conf. on Measurement and Modeling of Computer Systems*, Annapolis, MD, June 2008.
[27] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi. An analytical model for multi-tier Internet services and its applications. In *ACM SIGMETRICS Int'l Conf. on Measurement and Modeling of Computer Systems*, Banff, Canada, June 2005.