Configuration-Space Performance Anomaly Depiction

Christopher Stewart Kai Shen
Department of Computer Science
University of Rochester
{stewart,kshen}@cs.rochester.edu

Arun Iyengar Jian Yin
IBM Watson Research Center
{aruni, jianyin}@us.ibm.com

ABSTRACT
Complex distributed systems (like those based on J2EE platforms) are designed to perform well for a variety of application workloads and configuration settings. In practice, however, the system performance may not meet the expectation at all execution conditions. This short paper describes our research on depicting performance anomaly manifestations over a large execution condition space for distributed systems. Specifically, we provide a case examination of our research target and describe our work in three areas—performance anomaly identification, configuration-space anomaly depiction, and the utilizations of depiction results.

Categories and Subject Descriptors
C.4 [**computer systems organization**]: performance of systems—*modeling techniques, performance attributes*; C.5.5 [**computer systems organization**]: computer system implementation—*servers*; D.2.9 [**software engineering**]: management—*configuration management*; D.4.8 [**software**]: operating systems—*modeling and prediction, queuing theory*

General Terms
Performance anomaly, performance bug, system management, resource allocation, decision tree

Keywords
Performance Anomaly Depiction

1. PROBLEM DESCRIPTION AND RESEARCH GOAL
Performance anomalies, situations in which performance falls below expectations, are not uncommon in complex computer systems [1–3, 17, 21]. Causes for performance anomalies include overly simplified implementations, mis-handling of special/boundary cases, and improper management of system component interactions. Aside from their effect on performance degradation, these anomalies also make the system performance behaviors hard to predict. Many system management functions would benefit from such predictability. For instance, they can guide policy decisions on quality-of-service management and optimal resource provisioning [5, 8, 18–20, 23].

Our research investigates scalable techniques to examine a large space of potential system execution conditions and depict the conditions under which performance anomalies are likely to occur. Here by an *execution condition*, we mean the specification of a system running environment that includes system configuration parameters (*e.g.*, component placement policy and caching protocol) and input workload properties (*e.g.*, component CPU needs and inter-component communication patterns). Figure 1 illustrates an example of execution conditions and performance anomaly depiction of a multi-dimensional space.

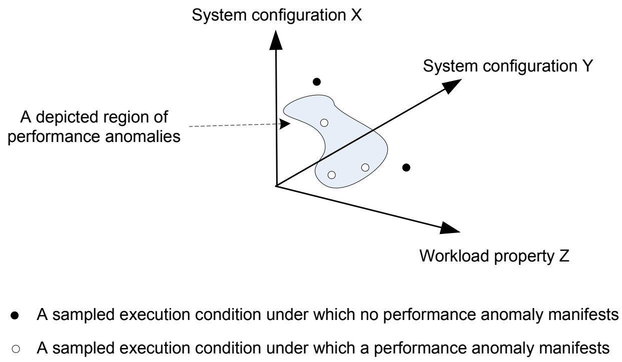

Figure 1: Execution conditions and performance anomaly depiction across a multi-dimensional space of system configurations and workload properties. The black and white dots represent sampled conditions where we have measured the actual performance to determine if a performance anomaly manifests. These sampled conditions are the basis for performance anomaly depiction.

The performance anomaly depiction provides a probabilistic view of anomaly manifestations under each potential execution condition. The results can be used to guide the avoidance of anomaly-inducing system configurations under given input workload properties. In particular, system configurations that are expected to meet performance requirements according to high-level system design may fail to do so when anomalies manifest under such configurations. Additionally, we can identify the execution conditions that are most correlated with anomaly manifestations. This information can help narrow the search space for performance debugging to the system functions that are affected by the problematic conditions.

In principle, the proposed anomaly depiction approach can be applied to any software system that allows many configuration settings and supports various application workloads. We are particularly interested in the case study on component-based distributed systems. A typical example of such systems may be built on a J2EE platform and may support various application components, provide common services (possibly remotely), and manage application-level protocols to access storage and remote services. Additionally, these systems are often deployed in distributed environments with high-concurrency workloads. Our current empirical studies employ an open-source J2EE distributed middleware platform (JBoss) as well as a commercial J2EE distributed middleware platform (IBM WebSphere).

Related Work

Prior research investigated techniques to discover behavioral anomalies or failure causes in large system software [6, 11]. However, the identification of performance anomalies is uniquely challenging because they typically relate to high-level system semantics while they do not exhibit easily identifiable system crashes, incorrect states, and source-level patterns.

The detection, characterization, and debugging of performance anomalies have been investigated in recent research [1–3, 14]. In general, these studies focused on anomaly-related issues under the specific execution conditions encountered during a particular execution. However, distributed systems often allow many configuration settings (*e.g.*, cache coherence protocol and thread pool size) and support workloads with varying properties (*e.g.*, service concurrency and database access rate). It is desirable to explore performance anomalies over a comprehensive set of execution conditions, which allows quality assurance under varying conditions and exposes anomaly-correlated execution conditions to help the root cause analysis.

There is a large body of previous work addressing fault tolerance and high availability of distributed system designs, such as distributed consensus [4, 10] and replication management [13, 16]. At a high level, our research does not propose new *design* techniques or principles for distributed systems, but instead it tries to discover *implementation* errors that cause deviations of system behaviors from the high-level design (called anomalies). Implementation deviations from design are not rare in distributed systems given their increasing complexity.

2. A CASE EXAMINATION

To gain a concrete understanding on our target problem, we provide a case examination of the large execution condition space and condition-specific performance anomalies for distributed systems.

System configurations	**Values**
Database connector	Buffered results, streaming results
EJB component cache coherence	No cache, exclusive-access, verification-based
Remote invocation	Java RMI, JBoss-specific
Component placement strategy	All components co-located (with web server, database, or neither), cpu-heavy comp. co-located (with web server, database, or neither), net-heavy comp. co-located (with web server, database, or neither), cpu load-balanced placement, net load-balanced placement
Invocation retry policy	Never retry, retry once
Maximum system concurrency (max web-server threads)	Small (10), medium (128), medium-high (512), high (2048)

Workload properties	**Values**
HTTP session type	HTTP 1.0, HTTP 1.1, SSL
Database access frequency in the workload request mix	0%, 25%, 50%, 75%, 100% (percentage of requests with more than one database access)
Type of hosted RUBiS components	Servlet only (no EJB components), EJB components with bean managed state persistence (BMP), EJB components with container managed state persistence (CMP), EJB components with session state, stateless EJB components
Workload request rate	1, 3, 6, 9, $\cdots$, 180 requests per sec. (61 distinct settings)

Figure 2: Some possible system configurations and workload conditions in a JBoss/RUBiS system. The component placement strategy affects which machines RUBiS EJB components run on. For instance, the "cpu-heavy co-located with web server" setting means that the EJBs with high CPU usage run on the same machine as the web server. The other components would run on the their own machine (*i.e.*, the "neither" machine).

Specifically, we consider a deployed system on the JBoss Application Server [9], the most widely used open-source J2EE platform. The whole system contains the JBoss Application Server, the Tomcat Servlet container and related plugins, MySql database with its JDBC connector, and the RUBiS online auction benchmark [15]. RUBiS is a three-tier web service comprising a web server, a database, and nine middle-tier Enterprise Java Beans (EJB). The various RUBiS application components are distributed across a three-machine cluster. The web-server is bound to one machine, the database to another, and the EJB components can be assigned to any machine. Figure 2 lists six J2EE configurations and four properties of the application workload. Different parameter values on each of the ten parameters allow about 4.8 million possible execution conditions for this system.

We describe an example performance anomaly scenario. In this example, we observe a degradation of system performance when an increasing proportion of the input requests trigger multiple database accesses (the "original performance" in Figure 3). The cause of this performance degradation is described below. The Servlet-only implementation of RUBiS bypasses the default JBoss database connection management and it instead manages database connections on its own. When a database connection is closed on the server side, RUBiS may not always properly restart the broken connection. Some future requests may then be assigned the broken connection and fail on database accesses, which manifests as a reduction of successful request completions. A simple correction is to check for broken database connections and reconnect them when necessary. Figure 3 shows that the correction can eliminate the anomalous performance degradation at high database access rates.

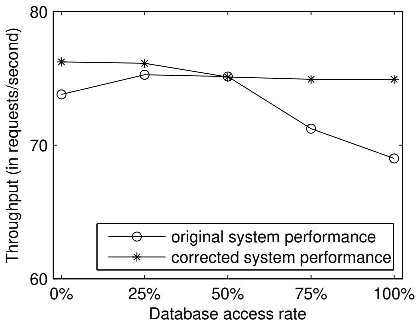

Figure 3: An example performance anomaly over certain workload configurations and its correction for the JBoss/RUBiS system. Here the database access rate of a workload indicates the percentage of requests with more than one database access. Note that the Y-axis does not start from zero.

3. RESEARCH ISSUES AND PRELIMINARY WORK

We describe several research issues and provide an overview of our preliminary results on these issues.

3.1 Performance Anomaly Identification

Literally speaking, a performance anomaly arises when the system performance behavior deviates from the expectation. Performance expectations can be made in different ways, such as design-driven models, service-level agreements, or programmer specifications. Our goal is to derive performance expectations that match high-level design principles and that can be intuitively interpreted. For instance, the expected performance of a distributed cache should match that intended by the high-level caching algorithm. Given a design-driven expectation, a performance anomaly would indicate an implementation deviation from the high-level design.

Performance expectations can be derived by following some (typically bottom-up) design semantics to assemble expectations from low-level system metrics that are acquired through offline calibration or online measurements. Past efforts (including others [7, 22, 23] and our own [18–20]) have constructed design-driven performance expectation models for distributed systems. It is worth noting that the IRONModel paper [22] has recognized the discrepancies between real system performance behaviors and traditional design-driven expectation models. They tried to incorporate these anomalies into the expectation models while our work attempts to better understand them (and particularly their manifestation patterns over the large execution condition space).

In our preliminary work, we derive performance expectations under each configuration based on our prior modeling work [20]. Specifically, we model multi-component network services by capturing the first-order resource consumption in individual component executions and inter-component communications. Given characterized application behavior profiles, our model allows us to derive performance expectations under different distributed component placement and resource provisioning policies. We are aware that a deviation from the high-level design expectations may also be due to measurement errors or natural behavior instability in complex systems, including effects of sporadic background maintenance like garbage collection and system event logging. Anomalies caused by these factors are usually small in magnitude. We may screen them out by only counting the relatively large performance deviations.

3.2 Configuration-Space Anomaly Depiction

From a representative set of sampled execution conditions (each labeled "normal" or "anomalous"), we derive performance anomaly depiction over a comprehensive execution condition space. The depiction is essentially a classifier of system configurations and workload properties that partitions the execution condition space into normal and anomalous regions. There have been a great number of well-proven classification techniques, such as naive Bayes classifiers, perceptrons, decision trees, neural networks, Bayesian networks, support vector machines, and hidden Markov models. We use decision trees to build our anomaly depictions because they have the desirable properties of easy interpretability, prior knowledge free, and robustness in handling noises.

Our depiction is a decision tree that classifies a vector of workload properties and system configurations into one of two target classes—"normal" or "anomalous". Guided by the Iterative Dichotomiser 3 algorithm [12], the decision tree is generated in a top-down fashion by iteratively selecting an attribute that best classifies current training samples, partitioning samples based on their corresponding values of the attribute, and constructing a sub-tree for each partition. At each step of the tree-generation algorithm, we choose the attribute most effective in classifying training samples according to the measure of *information gain*. Intuitively, information gain represents the amount of reduction on the co-existence of normal and anomalous conditions in one partition. Our preliminary results on anomaly depiction has been presented in [21]. Figure 4 illustrates an example performance anomaly depiction for a sub-space of the JBoss/RUBiS execution conditions.

3.3 Depiction Utilizations

Performance anomaly depictions are useful when configuring or re-configuring online J2EE services. Our depictions can identify anomaly-inducing configuration settings given the input workload to the system, which can then guide

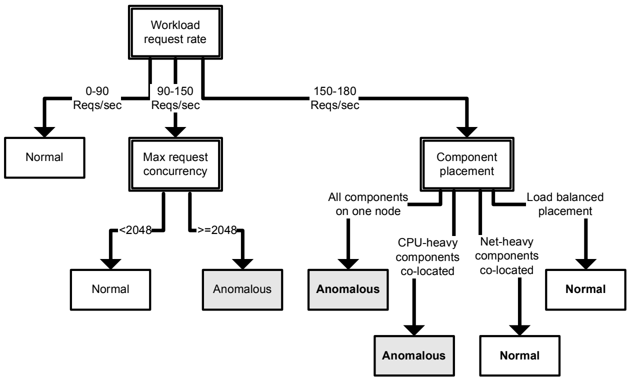

Figure 4: An example performance anomaly depiction in the form of a two-level decision tree. In this example, the top-level classifying attribute in the decision tree is the workload request rate. Two second-level attributes are the maximum request concurrency and the component placement policy.

the system to avoid such anomalous configurations. Specifically, our depiction results enable an approach to quantify the *performance dependability* of each configuration setting. Consider a system with possible configurations C=$\{c_1, c_2, \cdots, c_m\}$. The system users may exert a set of workload conditions W=$\{w_1, w_2, \cdots, w_n\}$ while the probability for condition w_i to occur is p_i ($\sum_{i=1}^{n} p_i = 1$). Our performance anomaly depiction indicates whether a specific workload condition will trigger an anomaly under a specific system configuration. In other words, the anomaly depiction is a function that maps $(C, W) \xrightarrow{f} \{\text{normal}, \text{anomalous}\}$. We may then assess a system configuration c's performance dependability using the probability that the system will perform normally under the full range of possible input workloads: $\sum_{1\leq i\leq n,\ (c,w_i)\xrightarrow{f}\text{normal}} p_i$.

We can also utilize performance anomaly depictions to aid performance debugging. Specifically, the depiction's internal structure may uncover correlations between execution conditions and performance anomalies. In the decision tree, each anomalous leaf node is correlated with the set of execution conditions present in its path from the root. Using system-specific expert knowledge, one can further link anomaly-correlated execution conditions to relevant system functions in the implementation. As an example, the performance anomaly described in Section 2 was discovered using our approach.

Acknowledgment

This work was supported in part by NSF grants CCF-0448413, CNS-0615045, CCF-0621472, and by an IBM Faculty Award. Ming Zhong (currently at Google) helped us in identifying the decision tree classifier for performance anomaly depictions.

4. REFERENCES

[1] M. Aguilera, J. Mogul, J. Wiener, P. Reynolds, and A. Muthitacharoen. Performance Debugging for Distributed Systems of Black Boxes. In *ACM Symp. on Operating Systems Principles*, pages 74–89, Bolton Landing, NY, Oct. 2003.

[2] M. Chen, A. Accardi, E. Kiciman, J. Lloyd, D. Patterson, A. Fox, and E. Brewer. Path-Based Failure and Evolution Management. In *USENIX Symp. on Networked Systems Design and Implementation*, pages 309–322, San Francisco, CA, Mar. 2004.

[3] I. Cohen, J. Chase, M. Goldszmidt, T. Kelly, and J. Symons. Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control. In *USENIX Symp. on Operating Systems Design and Implementation*, pages 231–244, San Francisco, CA, Dec. 2004.

[4] F. Cristian and C. Fetzer. The Timed Asynchronous Distributed System Model. *IEEE Trans. on Parallel and Distributed Systems*, 10(6):642–657, June 1999.

[5] R. Doyle, J. Chase, O. Asad, W. Jin, and A. Vahdat. Model-based Resource Provisioning in a Web Service Utility. In *USENIX Symp. on Internet Technologies and Systems*, Seattle, WA, Mar. 2003.

[6] D. Engler, D. Chen, S. Hallem, A. Chou, and B. Chelf. Bugs as Deviant Behavior: A General Approach to Inferring Errors in Systems Code. In *ACM Symp. on Operating Systems Principles*, pages 57–72, Banff, Canada, Oct. 2001.

[7] G. C. Hunt and M. L. Scott. The Coign Automatic Distributed Partitioning System. In *USENIX Symp. on Operating Systems Design and Implementation*, pages 187–200, New Orleans, LA, Feb. 1999.

[8] IBM Autonomic Computing. http://www.research.ibm.com/autonomic/.

[9] The JBoss J2EE Application Server. http://www.jboss.com.

[10] L. Lamport. The Part Time Parliament. *ACM Trans. on Computer Systems*, 16(2):133–169, May 1998.

[11] Z. Li, S. Lu, S. Myagmar, and Y. Zhou. CP-Miner: A Tool for Finding Copy-paste and Related Bugs in Operating System Code. In *USENIX Symp. on Operating Systems Design and Implementation*, pages 289–302, San Francisco, CA, Dec. 2004.

[12] J. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[13] R. Renesse and F. Schneider. Chain Replication for Supporting High Throughput and Availability. In *USENIX Symp. on Operating Systems Design and Implementation*, pages 91–104, San Francisco, CA, Dec. 2004.

[14] P. Reynolds, C. Killian, J. Wiener, J. Mogul, M. Shah, and A. Vahdat. Pip: Detecting the Unexpected in Distributed Systems. In *USENIX Symp. on Networked Systems Design and Implementation*, San Jose, CA, May 2006.

[15] RUBiS: Rice University Bidding System. http://rubis.objectweb.org.

[16] F. Schneider. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Computing Surveys*, 22(4):299–319, Dec. 1990.

[17] K. Shen, M. Zhong, and C. Li. I/O System Performance Debugging Using Model-driven Anomaly Characterization. In *USENIX Conf. on File and*

Storage Technologies, pages 309–322, San Francisco, CA, Dec. 2005.

[18] C. Stewart, T. Kelly, and A. Zhang. Exploiting nonstationarity for performance prediction. In *Second EuroSys Conf.*, Lisbon, Portugal, Mar. 2007.

[19] 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.

[20] C. Stewart and K. Shen. Performance Modeling and System Management for Multi-component Online Services. In *USENIX Symp. on Networked Systems Design and Implementation*, pages 71–84, Boston, MA, May 2005.

[21] C. Stewart, M. Zhong, K. Shen, and T. O'Neill. Comprehensive Depiction of Configuration-dependent Performance Anomalies in Distributed Server Systems. In *Second Workshop on Hot Topics in System Dependability*, Seattle, WA, Nov. 2006.

[22] E. Thereska and G. R. Ganger. IRONModel: Robust Performance Models in the Wild. In *ACM SIGMETRICS*, pages 253–264, Annapolis, MD, June 2008.

[23] 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*, pages 291–302, Banff, Canada, June 2005.