Performance Anomaly Detection and Bottleneck Identification

OLUMUYIWA IBIDUNMOYE, FRANCISCO HERNÁNDEZ-RODRIGUEZ,
and ERIK ELMROTH, Umeå University

In order to meet stringent performance requirements, system administrators must effectively detect undesirable performance behaviours, identify potential root causes, and take adequate corrective measures. The problem of uncovering and understanding performance anomalies and their causes (bottlenecks) in different system and application domains is well studied. In order to assess progress, research trends, and identify open challenges, we have reviewed major contributions in the area and present our findings in this survey. Our approach provides an overview of anomaly detection and bottleneck identification research as it relates to the performance of computing systems. By identifying fundamental elements of the problem, we are able to categorize existing solutions based on multiple factors such as the detection goals, nature of applications and systems, system observability, and detection methods.

Categories and Subject Descriptors: C.4 [**Computer-Communication Networks**]: Performance of Systems—*Reliability, availability, and serviceability*; D.4.8 [**Operating Systems**]: Performance—*Measurement, modeling and prediction*

General Terms: Performance, Reliability

Additional Key Words and Phrases: Systems performance, performance anomaly detection, bottleneck detection, performance problem identification

ACM Reference Format:
Olumuyiwa Ibidunmoye, Francisco Hernández-Rodriguez, and Erik Elmroth. 2015. Performance anomaly detection and bottleneck identification. ACM Comput. Surv. 48, 1, Article 4 (July 2015), 35 pages.
DOI: http://dx.doi.org/10.1145/2791120

1. INTRODUCTION

Modern enterprise applications and systems most often function well but are still known to sometimes exhibit unexpected and unwanted performance behaviours with associated cost implications and failures [Pertet and Narasimhan 2005]. These performance behaviours or anomalies are often the manifestations of bottlenecks in the underlying system. In fact, many factors such as varying application load, application issues (e.g., bugs and updates), architectural features, and hardware failure have been found to be sources of performance degradation in large-scale systems [Cherkasova et al. 2009; Magalhaes and Silva 2010]. Regardless of the sources of the problem, the challenge is how to detect performance anomalies and how to identify potential root-causes. The scale, dynamics, and heterogeneity of today's IT infrastructure further aggravate the problem.

This work is supported by the Swedish Research Council (VR) under contract number C0590801 for the Cloud Control project and the European Union' s Seventh Framework Programme under grant agreement 610711 (CACTOS).
Authors' addresses: Department of Computing Science, Umeå University, Umeå SE-90187, Sweden; emails: {muyi, elmroth, francisco}@cs.umu.se.

DOI: http://dx.doi.org/10.1145/2791120

Performance bottlenecks and anomalies are barriers to achieving predictable performance guarantees in enterprise applications and often come with significant cost implications. Studies [Kissmetrics 2014] have shown that there exist correlations between the end-user performance and sales or number of visitors in popular web applications and how consistently high page latency increases the page abandonment rate. It was also shown that for a small-scale e-commerce application with a daily sales of $100,000, a 1-second page delay could lead to about 7% loss in sales annually. Also according to Huang [2011], Amazon experiences a 1% decrease in sales for an additional 100ms delay in response time while Google reports a 20% drop in traffic due to 500ms delay in response time. These implications show not only the importance but also the potential economical value of robust and automated solutions for detecting performance problems in real time.

Similarly, if left unattended, performance bottlenecks may eventually lead to system failure and outages spanning minutes to weeks. Bottleneck conditions, such as system overload, and resource exhaustion have been reported to cause prolonged and intermittent system downtimes [Pertet and Narasimhan 2005]. Global web services such as Yahoo Mail, Amazon Web Services, Google, LinkedIn, and Facebook have recently suffered from such failures [McHugh 2013]. Unplanned downtimes have significant cost implications [Evolven 2011] not just in lost sales but also in man-hours spent on recovery. To achieve guaranteed service reliability, performance, and Quality of Service (QoS), timely detection of performance issues before they trigger unforeseen service downtime is critical for service providers [Guan et al. 2011]

Considerable efforts have been made to address this issue in the academia with interesting proposals. Many of these solutions leverage the power of statistical and machine learning techniques. Though many of these efforts have been concentrated on solving the problem in specific application domains, the characteristics of the problem and proposed solutions are similar. A basic performance anomaly detection and bottleneck identification (PADBI) system observes, in real time, the performance behaviours of a running system or application, collects vital measurements at discrete time intervals to create baseline models or profiles of typical system behaviours. It continuously observes new measurements for deviations in order to detect expected or unexpected performance anomalies and carry out root-cause analysis to identify associated bottlenecks. This survey aims at providing an overview of the problem and research on the topic. We provide a basic background on the problem with respect to the fundamental elements of the process, methods, and techniques while identifying research trends and open challenges.

1.1. Our Contribution

This work is an attempt to provide thorough description of the performance anomaly detection and bottleneck identification problem and to present the extensive research done in this area. The diverse nature of works addressing this problem informs this work, and we herein present our findings. A similar survey on the general problem of anomaly detection is presented in Chandola et al. [2009]. We start by giving a general background while identifying core elements of the problem. Then we discuss the main contributions of various authors organized in terms of the systems, goals, and techniques used. We conclude by discussing research trends, future directions, and specific requirements for Cloud computing.

1.2. Organization

We introduce the article in Section 1. Section 2 presents a background of the problem and discusses the concept of performance anomalies and bottlenecks, their root-causes, and other fundamental concepts. In Section 3, we address the various detection

strategies and techniques employed in existing literature. Section 4 summarizes past and present research trends while also describing specific Cloud computing requirements in Section 5. Section 6 discusses important concerns about detection methods and presents future directions in terms of challenges and open issues. We conclude in Section 7.

2. BACKGROUND

2.1. Basic Concepts

The performance of computer systems is typically characterized in terms of the duration taken to perform a given set of tasks or the rate at which these tasks are performed with respect to the amount of system resources consumed within a time interval [Gregg 2013].

Performance metrics are key performance indicators (KPI) derived from fundamental system measurements (such as count, duration, and size) to describe the state of operation of a computer system. The two most popular metrics are the *response rime* (or *latency*) and *throughput*. Latency is broadly used to describe the time for any operation to complete, such as an application request, a database query, a file system operation. The throughput of a system is the rate of work performed. For instance, in web applications, throughput is the number of users' requests completed within a time interval (e.g., requests or transactions completed per second) [Gregg 2013].

System *resources* includes physical components such as the CPU, memory, disk, caches, network and virtual components such as the network connections (e.g., sockets), locks, file handles or descriptors [Gregg 2013]. Resource *capacity* describes the storage size or processing strength of a given resource, such as the number of CPUs, and the size of physical memory or disk.

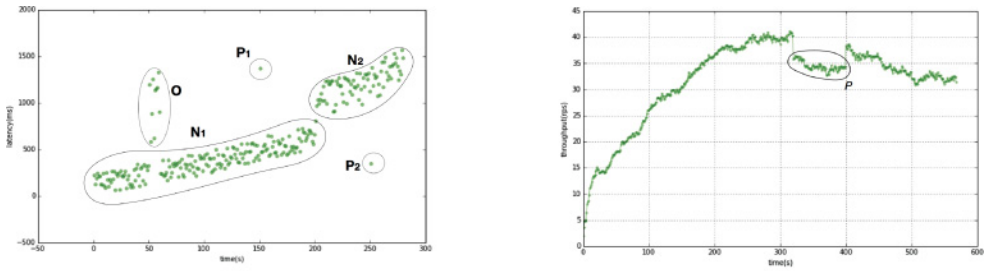
Resource *utilization* of an application typically captures the amount of capacity used with respect to the available capacity. For example, CPU usage is measured as the amount of time (in percentage) the CPU is busy executing instructions from an application, while memory utilization measures (in percentage) amount of storage capacity consumed by a particular process or application. Utilization of network resources may capture the ratio of number of packet transmitted to the full transmission capacity of a network link in a given time interval [Shallahamer 1995; Lilja 2005].

In the following sections, we present various aspects of the PADBI problem.

2.2. Performance Anomalies

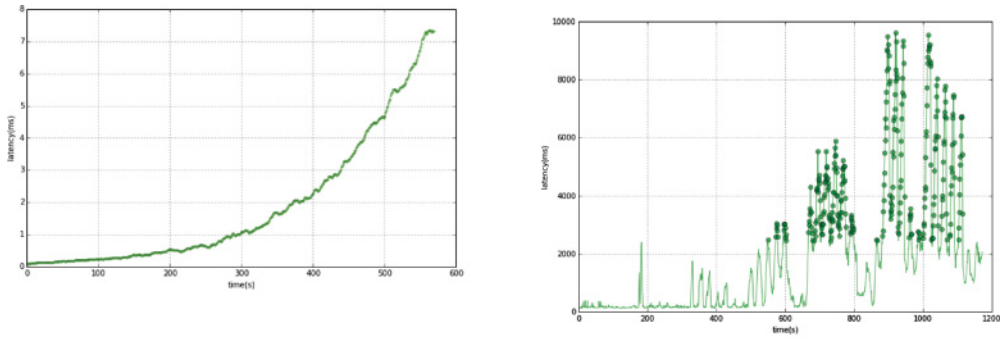
Generally, anomalies can be seen on a graph, as a point or group of data points lying outside an expected normal region [Das 2009]. In performance studies, the data points are discrete measurements of a performance metric, throughput, for example. Figure 1(a) is a plot of *latency* against *time* for an hypothetical system. The two homogeneous clusters (N_1 and N_2) represent the normal operating region, while the points (p_1 and p_2) or group of points (O) falling outside the normal regions are anomalies or outliers. Figure 1(b) captures another example of throughput anomaly, the group of points P represent a short dip in system throughput.

2.2.1. Types of Anomalies. Chandola et al. [2009] identify three basic types of anomalies: *point*, *collective,* and *contextual*. These types only capture anomaly in terms of individual or contiguous data points; however, performance metrics are also known to commonly exhibit characteristic shapes when a resource is saturated [Gunther 2004]. Therefore, we present one more type of anomaly, the *pattern* anomaly, which characterizes performance behaviours in terms of the structure or shapes of their curves rather than finite data points [Gunther 2011].



(a) Latency anomaly. (b) Throughput anomaly.

Fig. 1. Illustration of anomalies.



(a) Anomalous latency growth pattern. (b) Latency spikes.

Fig. 2. Latency anomalies.

(1) *Point anomalies.* A point anomaly is any point that deviates from the range of expected values in a given set of data. For example, a memory usage value 3 standard deviation from the mean (i.e. $>= \mu \pm 3\sigma$) may be considered a point anomaly if the expected behaviour is 1 standard deviation from the mean (i.e., $<= \mu \pm 1\sigma$). In Figure 1(a), points labeled p_1 and p_2 are point anomalies. Point anomalies are the dominant type of anomalies in majority of literature that we reviewed. They commonly manifest as spikes in application latency or system resource utilization measurements. Figure 2(b) shows a plot of application latency with respect to time. The solid dots indicates detected point anomalies.

(2) *Collective anomalies.* Collective anomaly is a homogeneous group of data points deviating from the normal regions of the rest of the data. Though the individual data points may not be anomalous with respect to the group, their occurrence together as a collection is anomalous. An unexpected streak of low-throughput values may be considered anomalous when compared with higher-throughput behaviour in past observation windows. In Figure 1, the group of points labeled O in Figure 1(a) and points labeled P in Figure 1(b) are collective anomalies.

(3) *Contextual anomalies.* Some performance anomalies manifest only under specific execution environments or contexts. The contexts may be defined by load levels (e.g., high, moderate, load, or bursty), type of payloads (e.g., IO-bound, CPU-bound, read-heavy, write-heavy or mixed), system states (e.g., system configurations), or by the nature of underlying computing infrastructure (e.g., virtualized or shared-hosting environments), and so on.

(4) *Pattern anomalies.* The shapes of some performance metrics when plotted are known to exhibit specific pattern that can be used to identify anomalous behaviours

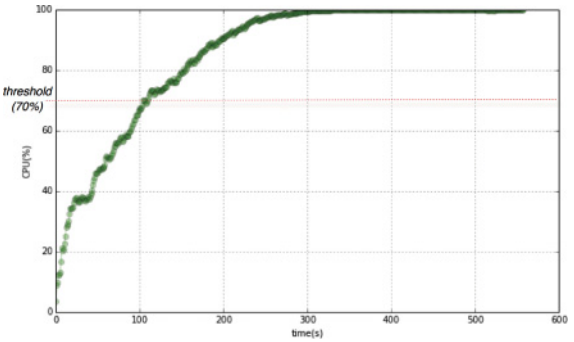


Fig. 3. CPU saturation bottleneck.

[Gunther 2011]. For example, application latency is known to exhibit an asymptotic growth as seen in Figure 2(a). The shape or pattern of performance metric may be anomalous if it does not conform to the shape of typical behaviour. Pattern anomalies may be considered a generalized form of collective anomalies because the anomalous shape is made up of a set of data points that are also collective anomalies.

2.3. Performance Bottlenecks

A bottleneck is a resource or an application component that limits the performance of a system [Gregg 2013]. Malkowski et al. [2009] describe a bottleneck component as a potential root-cause of undesirable performance behaviour caused by a limitation (e.g., saturation) of some major system resources associated with the component [Lee et al. 2012]. Such components often exhibit frequent congestion of load [Mi et al. 2008a]. Also, application or system metrics correlating with an observed performance limitation are referred to as bottleneck metrics [Parekh et al. 2006].

2.3.1. Types of Bottlenecks.

2.3.1.1 Resource Saturation Bottlenecks. A resource is saturated when its capacity is fully utilized or past a set threshold [Gregg 2013]. For example, Figure 3 depicts a saturated CPU past the threshold usage level of 70%. According to Gregg [2013], saturation may also be estimated in terms of the length of a resource queue of jobs or request to be served by that resource. Saturation causes different system resources to be bottlenecked differently with varying performance impact. CPU—near 100% utilization resulting in congested queue and growing latency. Memory—constrained capacity due to limited physical memory or deprivation caused by *memory leaks*[1] leading to constant paging and swapping. Disk Saturation—constant disk access beyond available bandwidth forcing new IO requests to queue up. Network saturation—network congestion due to fully utilized bandwidth causing new traffic to be delayed or dropped.

2.3.1.2. Resource Contention Bottlenecks. In multitasking environments, application processes contend for limited system resources such as CPU cycles, IO bandwidth, and physical memory, and also software resources such as buffers, queues, semaphores, and mutexes. The impact of such contention is well pronounced in cloud data centers due to resource interference between multiple cloud tenants. The *noisy neighbours* effect is an analogy for this interference [Pu et al. 2010]. Several contention scenarios are well known for different system resources: (1) CPU contention—multiplexing the CPU

[1]Memory leak is a classical memory bottleneck scenario where an application indiscriminately allocate memory spaces that are never deallocated thereby saturating the memory and starving other users.

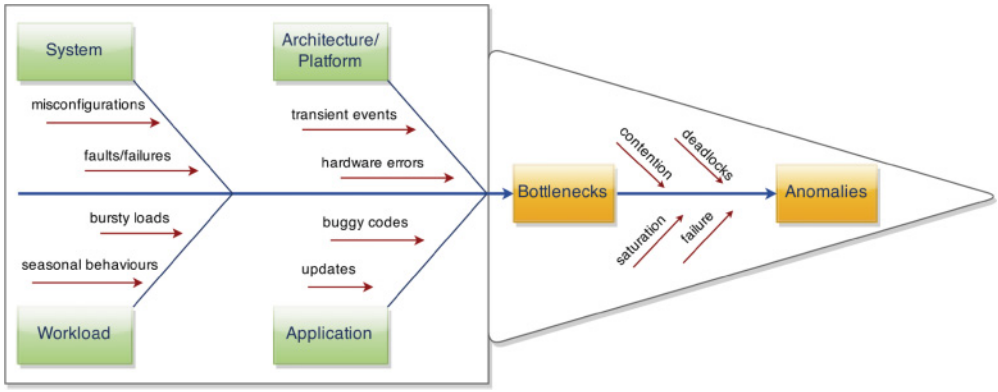


Fig. 4. Cause-effect relationships of performance anomalies and bottlenecks.

between multiple processes causes frequent congested queue and performance interference in virtualized systems especially in the presence of CPU *hogging*[2] programs. Memory contention—sharing limited memory bandwidth and processor-memory interconnect among processes may result in significant performance impact. (3) Disk Contention—the processor-IO performance gap and restricted disk payload[3] causing substantial performance loss especially in IO workloads. (4) Network Contention—excess demands for communication links at peak times lowers the effective bandwidth offered resulting in undesirable network contention delays.[4]

2.3.2. Bottlenecks Behaviours. Performance bottlenecks manifests in different ways depending on applications and systems.

(1) *Single Bottlenecks.* Single bottlenecks exhibit predominant saturation at a single resource or component. An inherent characteristic of the bottleneck component is a near-linear load-dependent growth in resource usage. Malkowski et al. [2009].
(2) *Multiple Bottlenecks.* Two or more system resources or component may saturate simultaneously, or concurrently due to interdependency. Malkowski et al. [2009] classifies multiple saturation behaviours as simultaneous, oscillatory, and concurrent depending on saturation frequency given the presence of another saturation.
(3) *Shifting Bottlenecks.* Shifting bottlenecks are a special case of multiple ones. Due to fluctuating loads and the cascading nature of web requests, an application may experience shifts in bottleneck between two or more application components—the *domino* effects—due to interdependency between them.

2.4. Sources of Performance Anomalies and Bottlenecks

Figure 4 is an extended *Fish-bone* diagram explaining the interrelationships between performance bottlenecks, anomalies and their causes. The green boxes on the left are the main categories of root-causes, the red horizontal arrows are example of primary causes that further explain each category. The orange rectangles on the right are the main effects of the primary causes. Secondary causes that further explain primary causes and effects are represented with red slanted arrows. The thick horizontal arrow,

[2]CPU hogging programs place excessive demand on compute resources thereby impacting the performance of other applications on the same host.
[3]Disk payload is in terms of size (in bytes) and number of IO requests (read/write) per second.
[4]Network contention delay is expressed as ratio of possible demand for a given network link to its maximum capacity.

the *spline*, depicts how primary and secondary causes can be used to explain main effects from left to right.

2.4.1. Application Issues. Application-level issues such as incorrect tuning, buggy codes, and software updates are examples of bottleneck sources [Kelly 2005a]. Incorrect application configuration and updates may introduce unexpected resource bottlenecks [Cherkasova et al. 2009].

2.4.2. Workload. Bursty application loads are characterized by periods of continuous peak arrival rates that significantly deviate from the average or expected workload intensity. Internet phenomena such as *flash-crowds*[5] culminate into workload burstiness [Mi et al. 2008a]. The undesirable effects of such load behaviour include congested queues, oversubscribed threading resources, present of short and uneven peaks in resource and performance measurements [Casale et al. 2009].

2.4.3. Architectures and Platforms. Transient events such as those introduced by underlying system architecture or operating systems (e.g., memory hardware errors), occur over short timespan. Multiple occurrence of such transient errors and events results in bottlenecks that are hard to detect. Wang et al. [2013a] and Mi et al. [2008a] have shown how JVM garbage collection and Intel SpeedStep technology can induce bottlenecks. In modern systems with multicore NUMA architecture, the location of memory relative to a processor may affect application performance especially those with memory-bound workloads [Panourgias 2011].

2.4.4. System Faults. Faults in system resources and components may considerably affect application performance [Muppala et al. 1991] with significant cost. System failures may be intermittent, transient. or even permanent. Reasons for such failures can be attributed to software bugs, operator error, hardware faults, environmental issues, and security violations. In recent times, many popular web application services have been hit by failures that temporarily disrupt their application services for some time [Pertet and Narasimhan 2005].

2.5. Core Elements of the Problem

2.5.1. Nature of the Problem. The complexity of today's systems makes the process of detecting performance issues and identifying root-causes nontrivial. We identify the following as the major challenges:

(1) *Dynamic Dependency*. At scale, applications comprise of multiple interdependent components deployed in data centers servers with heterogeneous and equally interdependent resources. This dependency results in dynamic behaviours. For example, hard-to-detect alternating or cascading bottlenecks between two or more components and resources is very common in large datacenters [Wang et al. 2013b].
(2) *Dynamic Anomaly Characteristics*. Today's systems are by nature highly dynamic with characteristic unpredictable behaviours. It follows that defining a priori all possible behaviours (normal or anomalous) of an application is technically unrealistic. Similarly, the notion of normality, anomalies and their characteristics vary widely across applications, execution environments, and load contexts. Therefore, it is hard to precisely distinguish normal application behaviours from anomalous behaviours at runtime [Lan et al. 2010].
(3) *Nature of Data*. There exist a diverse set of data collection tools, each generating output data in different formats and semantics. This makes it difficult to consume

[5]A flash-crowd is an Internet phenomenon where a network suddenly receives a huge influx of traffic due to breaking news, major events, natural disasters, and so on.

Table I. Recent Literature by Goals

PAD	PBI	PADBI
Zhang et al. [2007b]	Chen et al. [2002]	Cohen et al. [2004]
Gunter et al. [2007]	Malkowski et al. [2007]	Kelly [2005a]
Yang et al. [2007]	Chung et al. [2008]	Jung et al. [2006]
Cherkasova et al. [2008]	Ben-Yehuda et al. [2009]	Agarwala et al. [2007]
Lan et al. [2010]	Iqbal et al. [2010]	Kandula et al. [2009]
Fu [2011]	Xiong et al. [2013]	Malkowski et al. [2009]
Sambasivan et al. [2011]	Wang et al. [2013b]	Magalhães and Silva [2011]
Tan et al. [2012]		Magalhaes and Moura Silva [2011]
Pannu et al. [2012]		Lee et al. [2012]
Wang et al. [2012]		Kang et al. [2012]
Guan and Fu [2013b]		Dean et al. [2014]
Yu and Lan [2013]		
Sharma et al. [2013]		
Huang et al. [2013]		
Wang et al. [2014]		

the data in a uniform manner [Lan et al. 2010]. Also, due to the influence of varying data collection mechanisms, processing and transmission errors, performance data may suffer from the presence of noise whose values may be similar to anomalies. This complicates the detection problem, as noise often masquerades as anomalies resulting in high false-positive detections.

Furthermore, today's systems generate huge quantity of health or operational data that can easily overwhelm analysis and detection process. Finding anomalies and bottleneck symptoms in such datasets is analogous to finding a needle in a haystack.

2.5.2. System Goals. The general research questions behind PADBI systems are:

(1) *How to automatically detect anomalous performance behaviours?*
(2) *How to automatically identify the root cause of an observed performance anomaly?*
(3) *Which system resource or component is responsible for an observed violation of a performance objective?*

We refer to the goal of systems addressing the first question as *performance anomaly detection* (PAD). Systems addressing the second and third question are classified as *performance bottleneck identification* (PBI). In many cases, we observed a blurred line between papers addressing anomaly detection and those addressing bottleneck detection. We categorize such systems (i.e., addressing all the three questions) as *performance anomaly detection and bottleneck identification* (PADBI).

Table I classifies recent literature according to their goals, with PAD, PBI and PADBI systems accounting for 53%, 29% and 18% respectively of all literature reviewed as presented in Table IX and X of Appendix A.

Generally, the output of PADBI systems may include a set of anomalous performance *indices*, a *timestamp* (time of incident), a set of *anomalous metrics*, a *label* (in case of learning based systems)—an assigned class to which a sample belongs e.g. *normal* or *anomaly*, and an *anomaly score*—the degree to which a case is considered anomalous. The performance of PADBI systems themselves and their sensitivity are evaluated based on the following metrics:

(1) *Precision.* This is the ratio of correctly detected anomaly to the sum of correctly and incorrectly detected anomalies. It is also referred to as the *positive predictive value* (PPV) in literature.

(2) *Recall.* Also known as the *confidence score* or *true positive* (TP) rate. Recall is the ratio of correctly detected anomalies to all anomalous instances in a given dataset. It may be referred to as *densitivity* in some literatures. Conversely, the ratio of correctly detected normal instances to the total count of normal instances in the dataset is called *specificity* or the *true negative* (TN) rate.
(3) *Accuracy.* This is the ratio of the count of all correct detections (anomalies or not) to the total number of cases in the system.

2.5.3. Systems. PADBI have been studied in many system domains and application architectures. These include distributed applications (such as web-based client-server and multitier application) deployed in dedicated and shared server environments. Distributed applications are composed of highly specialized application entities integrated to achieve some high-level system objectives [Peiris et al. 2014]. While the Grid [Yang et al. 2007] is known for running short- and long-term applications performing large computations across distributed nodes, cloud infrastructures [Gong et al. 2010] allows diverse applications to share virtualized system resources (storage, compute, and network). The complexity of a system can be estimated by the number of resources and the composition of its applications.

System Resources. Resource demands are essential indicators of performance problems. The number of resources determines the size of the metric space and the volume of data that are eventually gathered. The interdependencies between system resources enables faults to propagate the system in a cascade manner (e.g. Disk and CPU resources).

Application Components. Modern applications are composed of heterogeneous software components distributed across separate and often geographically dispersed physical servers. In virtualized environments, application components are deployed in virtual machines (VMs) that can be migrated from one physical node to another within and across data centers. This complex composition and deployment brings special requirements for localization of performance problems.

Table II is a classification of research contributions according to system and application domains addressed.

2.5.4. Data. Performance data are a time series of the values of a set of performance metrics, systematically sampled over a regular interval. In this section, we briefly outline important aspects of such data.

Characteristics of Performance Data. Performance data are quantitative in nature. A performance metric is an attribute of a system or its component parts defining a state of the system. In general terms, metrics may also be referred to as *features* in some literature. A *case* or *instance* is a set of closely related features—a vector capturing a particular state of system at a point in time.

Sources of Performance Data. The bulk of performance data come from extensive measurements of metrics at two levels. *Application metrics* are foreground or in-band metrics that captures the current state or health of an application. Examples include application *response time* and *throughput*, number of *application users,* and *database connections*. *System metrics* are background or out-of-band metrics that capture the current state of the underlying system. Background metrics encompass not only resource utilization metrics but also hardware counters and error events. Examples are *CPU utilization*, number of *IO read/write* requests, *IO wait time*, and *CPU queue length.*

2.5.5. Data Collection. Monitoring is used to observe the runtime performance of a system by collecting both application- and system-level metrics using automated

Table II. Recent Literature on Systems

Reference	Dedicated Server	Virtualized, Cloud	Grid	Distributed	Web	Multitier
Wang et al. [2014]				✓	✓	✓
Dean et al. [2014]		✓				
Peiris et al. [2014]				✓		
Xiong et al. [2013]		✓		✓	✓	✓
Guan and Fu [2013a]		✓		✓		
Wang et al. [2013]		✓			✓	
Yu and Lan [2013]				✓		
Nguyen et al. [2013]		✓		✓	✓	✓
Sharma et al. [2013]		✓		✓	✓	✓
Tan and Adviser-Gu [2012]		✓		✓	✓	✓
Dean et al. [2012]		✓		✓		
Casale et al. [2012]		✓				✓
Fu et al. [2012]		✓		✓	✓	
Rathfelder et al. [2012]	✓			✓	✓	
Kang et al. [2012]		✓		✓		✓
Magalhaes and Moura Silva [2011]	✓			✓	✓	✓
Yu et al. [2011]		✓		✓	✓	✓
Do et al. [2011]		✓		✓	✓	✓
Lan et al. [2010]	✓					
Ben-Yehuda et al. [2009]		✓			✓	
Kandula et al. [2009]		✓		✓		
Gu and Wang [2009]	✓			✓		
Mi et al. [2008a]		✓		✓		✓
Gunter et al. [2007]			✓	✓		
Yang et al. [2007]			✓	✓		
Malkowski et al. [2007]	✓			✓	✓	✓
Agarwala et al. [2007]	✓			✓	✓	✓

third-party tools or via built-in Kernel counters. The efficiency of the detection process is influenced by three major aspects of data collection that are discussed next.

System Observability: White, gray or black box? The observability property of an application is greatly dependent on the type of infrastructure. In dedicated cluster environment, administrators have access to both application source codes and underlying infrastructure (*white-box*), such that both profiling and deep source tracing are possible possible. Whereas in cloud environments, cloud providers see applications as *black-boxes* while service providers (application owners) lack a global view of the infrastructure outside of their VMs. In general, white-box and gray-box systems allows for full and partial source code instrumentation, respectively. Such modifications are generally intrusive with significant runtime overhead. Black-box applications expose detail visibility into the application, thus limiting the amount of insights achievable but they are profiled in a nonintrusive manner [Nguyen et al. 2013].

Profiling vs. Tracing. Profiling extends beyond logging the state of system to studying the resource consumption behaviour and dependencies in order to assess the overall performance of the system. It also involves establishing analytical models that may be used to describe the dynamics of the system and predict performance [Shende 1999]. Examples of popular profiling tools include ps, *sysstat*, *htop*, *top*, *collectd*, *Nagios*, *Ganglia*, *apachetop*, *dstat*, and *iftop*.

Tracing is used to track fine-grained network or source-level events and misbehaviour via source code instrumentation. In addition to tracking the occurrence of certain events, tracing may reveal the execution flow, actions performed, caller-thread, and time spent in specific code blocks [Passing 2005]. Runtime code instrumentation is a common tracing method. Tracing platforms such as Aspect Oriented Programming [Tarby et al. 2007] and Java Byte Code instrumentation [Lee and Zorn 1997; Binder et al. 2007] have been used to observe applications. Examples of third-party tracing tools are *KProbe*, *AspectJ*, *JimysProbe*, *Dtrace*, *Magpie*, and *Strace*.

Influence of sampling interval. The volume of data generated by monitoring depends not only on metric space but also the rate at which we collect them. Shorter sampling intervals give finer resolutions than longer ones with additional compute and storage overheads. Longer sampling intervals produce lighter data but may miss out on transient performance events. An adaptive and selective monitoring is proposed in [Magalhaes and Moura Silva 2011]. The technique begins with a baseline sampling interval and continuously adjust the interval on the fly to adapt to the changing application behaviour. Also, metrics may be sampled selectively on demand.

3. SOLUTION STRATEGIES AND METHODS

Conventionally, the approach to detecting performance problems involves continuous estimation of models of normal system behaviours at specific points of interest. New performance observations that fail to match (within some acceptable confidence levels) existing models are flagged anomalous and system administrators alerted accordingly [Sharma et al. 2013]. However, many solutions employ more complicated techniques (such as statistical and learning methods) while following one or more strategies to achieve some detection goals. Different detection strategies and techniques are presented in Sections 3.1 and 3.2, respectively.

3.1. Detection Strategies

Existing PADBI systems often follow one or more strategies for robustness. A strategy defines the set of policies to achieve a detection goal. The choice of strategy is greatly influenced by system observability as well as whether the detection is to take place in either *offline* or *online* mode. Offline detection is a "post-mortem" identification and analysis of performance issues. Online detection is performed at run-time.

All strategies use thresholding to prune and complement detection decisions in one way or the other. A threshold is a limit value or range of values for parameters or metrics of interest beyond which an event is raised. Example thresholds include the p-value and R^2 (coefficient of determination) in statistical detection, distance from centroid (clustering), and entropy bounds (information theoretic) in machine learning detection. Setting thresholds becomes cumbersome when many parameters and metrics are involved. Modern system exhibit dynamic behaviours that consistently violates the ideal set thresholds. It is therefore expedient that the right thresholds are estimated. Threshold values are also expected to evolve with respect to change in underlying execution environment. In addition, it is crucial to understand the sensitivity of varying thresholds on detection accuracy.

Based on existing literature, we have identified four important strategies presented in Sections 3.1.1 through and 3.1.4. References of research contributions in each category can be found in Tables IX and X of Appendix A.

3.1.1. Signature-Based Detection. Applications exhibit specific behaviours at runtime that characterizes their performance such as their resources utilization, their performance, and load saturation rates. Such runtime characteristics are called *signatures*, *fingerprints,* or *profiles*. PADBI systems generate signatures as compact runtime

representations of important performance behaviours of an application. A signature may capture a normal system state or a deviation from that anomaly signature. Signature-based detection is a data-driven approach that consumes output of application profiling or tracing. Baseline signatures of prevailing system behaviour are generated in real-time and are then used to filter new observations for unwanted behaviours. An important attribute of signature-based detection is that signatures are discovered at runtime and may not require that a signature history is maintained. Generally, signature-based strategies require domain knowledge of and global snapshot of system state to achieve high accuracy. They usually record low false-positive detection and are suitable for known anomalies [Bodik et al. 2010; Bodík et al. 2008; Mi et al. 2008b; Cohen et al. 2005; Cherkasova et al. 2009].

3.1.2. Observational Detection. Applications can be observed through direct experimentation, staging (usually in a controlled environment) followed by in-depth analysis of observed anomalies and root-cause identification. To collect data, applications are either profiled in a black-box manner or source code instrumented for tracing. This approach covers both real-time analysis and "post-mortem" analysis of log files to discover sources of problems. Observational detection may also involve the staging of applications and systems where faults, anomalies, and bottlenecks are deliberately injected in order to understand system behaviour under such conditions. This approach is beneficial in various ways. First, it helps to prevent the limitation of hasty assumptions found in systems based only on analytical and simulation models. Second, it enables the understanding and identification of intrinsic behaviours of a system. Because this approach depends mostly on experiential and cognitive knowledge, it yields high accuracy in detecting known and unknown anomalies. This however, makes it difficult to implement in real-time situations because the experiential knowledge of right thresholds, transient anomaly behaviours have to be encoded into an automatic mechanism [Pu et al. 2007; Magalhaes and Moura Silva 2011; Tan et al. 2010].

3.1.3. Knowledge-Driven Detection. In specific enterprise systems, performance issues are often periodic with known root-causes and potential remedies. Many research and industrial systems leverage on such known anomalies and bottleneck definitions to identify and address performance problems. Knowledge-based detection approach identifies performance issues and their causes based on historical records of previously observed anomalies. It maintains a dynamic store (knowledge base) where definitions of known anomalies, their possible root-causes are maintained. The detection of new issues often trigger an update of the knowledge base. These definitions are converted into a set of formal rules that can be manipulated by an inference engine to detect performance issues and identify the root-causes. Although there exists some similarity between knowledge-based and signature-based detections, generation of rules and definitions does not necessarily have to be entirely at run-time in the former. This contrasts the online generation of signatures in the latter and does not require specialized inference engines. Knowledge-based detection is typically a data-driven approach and also require a great deal of understanding of the application and system domains. This strategy have high true-positive detection of known performance issues [Chung et al. 2008; Koehler et al. 2011; Li and Malony 2006].

3.1.4. Flow and Dependency Analysis. By studying the flow of communication across components in distributed applications, performance anomalies and hotspots can be easily identified. This approach typically involves real-time collection and analysis of traffic data (such as SNMP and TCP packets). In black-box systems, in-bound and out-bound network traffic may be observed to understand the performance behaviour of specific application components. Similarly, in white- or gray-box environments, dynamic code

tracing may be used to trace application requests or method invocations across code segments or network boundaries to better understand performance issues, their contexts and to pinpoint their sources. To detect anomalies and their causes, frequency, correlation, and causal path analysis are usually performed. Of great concern is the potential data collection overhead involved in this approach especially in large-scale systems with hundreds of applications and components. This approach often yield fine-grained detection with high true positives in black-box environments. Conversely, observing and understanding in-out traffic of hundreds of black-box components without prior knowledge may yield high false-positive detections [Ben-Yehuda et al. 2009; Sambasivan et al. 2011; Agarwala et al. 2007; Aguilera et al. 2003; Nguyen et al. 2013].

3.2. Detection Methods

To detect performance anomalies and identify associated bottlenecks, methods from diverse fields have been used, prominently from the domain of statistical analysis, machine learning (ML). We focus our discussion on statistical and learning techniques due to the volume of literature on them. However, signal processing methods such as *Extended Window Averaging*, *Adaptive Filtering,* and *Fourier Transforms* have also been used in Yang et al. [2007] and Malkowski et al. [2009]. We describe the commonly used techniques in literatures along with relevant references in Sections 3.2.1 and 3.2.2.

3.2.1. Statistical Detection. Statistical techniques provide capabilities to detect trends or drifts in critical performance metrics. Typically, researchers and system administrators observe system behaviours over time to make sense of underlying system dynamics. They construct models to hypothesize their observations, and employ some methods to estimate key model parameters and the relationship between them. Many statistical methods assume that some characteristics of the data are known a priori or can be inferred. For example, assuming the probability density of a performance metric follows a Gaussian (normal) distribution. These are called *parametric* statistical techniques. Examples of such methods are *Tukey limits*, *ANOVA* tests, *Pearson correlation*, *Grubb's Maximum normed residual,* and the *Student-t* tests. Nonparametric methods also exist that require little or no assumptions about the underlying nature of the data. Instead of assuming distribution of data as Gaussian, methods such *Histogram* or *Kernel* functions are used to estimate data distributions [Chandola et al. 2009; Rajasegarar et al. 2008]. The *Median*, *CUSUM*, *Spearman correlation*, *Kruskal-Wallis,* and *Wilcoxon's* tests are examples of nonparametric statistics [Burke 2001].

In general, statistical analysis provide a strong theoretical basis for detecting, and quantifying the influence of anomalies and bottlenecks on system performance. The assumption that the distribution of data is known a priori in many cases qualifies them for identifying well-known anomalies. However, many statistical methods exhibit sensitivity to variation especially when assumptions about the distribution of the data do not hold.

3.2.1.1. Gaussian-Based Detection. Gaussian-based techniques generally exploit the assumption that underlying data distribution is normal. Such techniques build Gaussian models parameterized by the mean μ, and variance σ^2 (i.e., $X \sim N(\mu, \sigma^2)$) [Chandola et al. 2009; Markou and Singh 2003].

The Tukey [1977] limits detect anomalous data points based on the distance from the distribution mean. The lower and upper normal thresholds are set at $(Q_1 - k * IQR)$ and $(Q_3 + k * IQR)$, respectively, where Q_1, Q_3 and IQR (computed as $Q_3 - Q_1$) are the 1st quantile, 3rd quantile, and the interquantile range, respectively. Data points outside this range are flagged anomalous. Though the threshold limit k is by default

1.5, it can be set to an appropriately chosen scalar for specific application [Wang et al. 2011].

The density distribution may also be exploited for detecting anomalous data points based on the Gaussian Mixture Model (GMM) [Markou and Singh 2003]. GMMs are parametric models of the probability distribution of continuous random variables estimated using the iterative Expectation-Maximization (EM) algorithm [Reynolds 2009].

Given a dataset X composed of n normally distributed features $\{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$ with corresponding parameters $\{\mu_1, \mu_2, \ldots, \mu_n\}$ and $\{\sigma_1^2, \sigma_2^2, \ldots, \sigma_n^2\}$, the Gaussian probability density function (PDF) for each feature $x^{(i)}$ is defined as $P(x^{(i)}; \mu_i, \sigma_i^2) = \frac{1}{\sigma_i\sqrt{2\pi}} \exp(-\frac{1}{2}(\frac{x^{(i)}-\mu_i}{\sigma_i})^2)$. The detection procedure proceeds first by estimating parameters μ_i and σ_i for each feature x_i. A new observation of the form $X = \{x^{(1)}, x^{(2)}, \ldots, x^{(n)}\}$ is classified anomalous if $P(X) < \epsilon$, where $P(X)$ is the sample's probability of being normal. The combined PDF of the dataset $P(X)$, is estimated as the product of the PDFs of each feature, that is, $P(X) = \prod_{i=1}^{n} P(x^{(i)}; \mu_i, {\sigma_i}^2)$. The value of ϵ can be varied depending on application requirements [Hodge and Austin 2004; Walck 2007].

Other density-based methods include the *Parzen* Windows Estimation [Parzen 1962], the *Grubb's* test [Grubbs 1969] and the *Student's t*-test [Markou and Singh 2003].

3.2.1.2. Regression Analysis. Regression analysis is a methodology for investigating relationships between performance metrics and to quantify the statistical significance of such relationships. For instance, regression analysis may explain how variation in load influences a given KPI with the assumption that the relationship (linear or nonlinear) between them is known a priori. The goal of regression is to estimate the set of model parameters that minimizes the absolute or the squared error. Commonly used algorithms for estimating model parameters include *Ordinary Least Squares (OLS)*, *Least Angle (LA)* and *Recursive Least Square (RLS)* [Kleinbaum et al. 2013].

For example, given a linear model of the form $T = \alpha(U_{cpu})$ describing the relationship between throughput and CPU utilization, a regression-based PADBI system first fits this model on a training data to estimate parameter α, the standardized p value, and the coefficient of determination (R^2). And for each test instance, the model computes the residuals—the variability in the test instance not explained by the model. The magnitude of the residuals are used to determine an anomaly score [Chandola et al. 2009]. New observations falling outside the confidence interval produced by the model may be classified as anomalous [Courtois and Woodside 2000; Lee and Brooks 2006]. An interesting use of regression models in modeling enterprise web applications is in generating *Transaction Mix* (TM) models. These models are used to describe application performance as a function of the mix of transactions (or requests) processed per unit time and their corresponding resource utilization. They are generally used for capacity planning and detecting transaction performance problems. Table III outlines literatures based on regression models.

3.2.1.3. Correlation Analysis. Correlation quantifies the degree of association between performance metrics. The interdependency of variables are estimated as a coefficient R in the range -1 to $+1$. Positive R values indicates a trend of increase in one variable as the other increases. Negative R values is a trend of decrease in one variable as the other increases. Variables sharing no association have R values of 0.0. Commonly used algorithms to estimate R are; the *Pearson*, the *Kendal* rank and the *Spearman* correlation [Magalhaes and Silva 2010]. Lets look at a simple example of how correlation may be used for detecting anomaly behaviour. Assume the correlation coefficient between two performance metrics A and B have been estimated in the confidence interval $[R_{min}, R_{max}]$ based on some training datasets. New observations of A and B over a time

Table III. Literature on Regression-Based Approaches

Author	Methodology
Kelly [2005a]	Presents a Regression-based TM model for identifying performance anomalies in geographically distributed applications. Tested with commercial applications such as ACME, FT, and VDR.
Zhang et al. [2007a]	Proposes an approach using Nonnegative Least Square (NLS) regressive TM models for estimating resource demands by different client transactions and applicability in resource provisioning.
Cherkasova et al. [2008]	Presents signature-based anomaly detection built on Zhang et al. [2007a]
Cherkasova et al. [2009]	Demonstrates how stepwise linear regression addresses model "overfitting" problem in Zhang et al. [2007a] and further present a segmentation-based method (as an extension of Cherkasova et al. [2008]) for detecting performance changes in enterprise web applications.
Kang et al. [2012]	Present a Regression-based diagnostic framework for analyzing performance anomalies and potential causes of SLA violations in virtualized systems. Their approach is based on Lassio, a variant of the Least Angle Regression (LAR) algorithm to identify suspicious system metrics accounting for observed performance anomaly.
Yang et al. [2007]	Models the relationship between application metrics and system metrics for metric selection, reduction, and anomaly detection.

window $[t_1, t_n]$ in the form $W_a = \{a_1, a_2, \ldots, a_n\}$ and $W_b = \{b_1, b_2, \ldots, b_n\}$ are anomalous if the R value between W_a and W_b falls outside the expected range $[R_{min}, R_{max}]$.

Canonical correlation is an advanced method that has been demonstrated for finding the linear association between one or more performance metrics. Given a vector of metrics $X = \{x_1, x_2, x_3, \ldots x_n\}$ and $Y = \{y_1, y_2, y_3, \ldots y_n\}$, the method computes the canonical variates (u, and v), the orthogonal linear combination of X and Y, that best capture the variability within and between X and Y. Kernel canonical correlation [Huang et al. 2006] is a popular variant of this method. See Table IV for references relating to correlation-based systems.

3.2.1.4. Statistical Process Control. Statistical process control (SPC) [Oakland 2008], is a quality control method widely used to monitor production processes for early detection of undesirable variation in process output. SPC provides a set of control charts, such as CUSUM, Shewart (ImR or XmR) charts, for monitoring process stability and variation. According to Bereznay and Permanente [2006], SPC is not suitable for interval based sampling data such as system performance traces. This motivates the development of the *Multivariate Adaptive Statistical Filtering (MASF)* method. MASF, Buzen and Shum [1995] is a SPC framework for detecting changes in a Gaussian distribution. MASF uses parameters mean (μ), standard deviation (σ), and variance (σ^2) of data collected during normal system operations as the basis for filtering subsequent system measurements for anomalous behaviours. For example, a MASF-based detection policy may set a control limit (CL) at the mean μ, a upper control limit (UCL) at $(\mu + 3\sigma)$ and a lower control limit (LCL) at $(\mu - 3\sigma)$. These control limits describe the range of expected variability in the data over a period of time. When new observations fall outside outside the set control limits, they are detected as anomalies and their cause(s) must be identified and corrected [Wang et al. 2011].

3.2.1.5. Statistical Intervention Analysis. Statistical Intervention Analysis (SIA) [Box and Tiao 1975] measures the form and magnitude of shifts in timeseries data. The shift is considered a consequence of a change, an *intervention* or *shock* in the data. It is particularly useful for studying the impact of an interventions (e.g., change in policy, natural disaster, or breaking news reports) on the behaviours of physical systems. In

Table IV. Literature on Correlation-Based Approaches

Author	Methodology
Agarwala et al. [2007]	Presents a performance management system where correlation analysis is used to identify important performance metrics and estimate the influence of specific application services and system resources on such them.
Magalhaes and Silva [2010] and Magalhaes and Moura Silva [2011]	Proposes an approach to identify potential root-causes of observed performance variation as either due to workload change or application update by computing the Pearson coefficient of correlation between aggregated workload, latency, and system metrics over some time window.
Kang et al. [2012]	Presents a method using correlation analysis to selecting model parameters. By filtering metrics showing collinearity relationships above a set threshold, they are able to reduce the dimension of models for detecting performance anomalies in virtualized infrastructures.
Sharma et al. [2013]	Uses correlation analysis to identify variations in performance metrics in a cluster of Virtual Machines (VM). They also show a technique to characterize anomalies by defining anomaly signatures in terms of changes to correlation between VMs in the cluster.
Wang et al. [2013] and Do et al. [2011]	Presents a Kernel-based Canonical correlation method to discover the correlation between workloads and performance in Internetware and how this is used for anomaly detection.
Gambi and Toffetti [2012]	Presents a novel Kriging-based model of system performance as a function of dynamic resource allocation and workloads to predict and detect performance problems.
Peiris et al. [2014]	Proposes a correlation-based method for automatic identification of associations among performance counters in a distributed system and how the association is used for detecting anomalies.

Table V. Literature on other Statistical Methods

Method	Highlight	Reference
ANOVA	Root-cause identification, and Change detection	Magalhães and Silva [2011] and Bereznay and Permanente [2006]
Index of Dispersion	Workload burstiness and variability detection in web applications	Mi et al. [2008a] and Casale et al. [2012]
Mean Standard Deviation, Cummulative Density Function	Anomaly detection in Grid application	Gunter et al. [2007]
Student's t-test	Localization of anomalous metrics	Wang et al. [2014]
Markov Model	Prediction of anomalous performance metrics and fault localization	Nguyen et al. [2013], Tan et al. [2012], Tan and Gu [2010], and Gu and Wang [2009]
Kernel Density Estimation	Estimation density functions using Kernel regression for inferring resource saturation	Malkowski et al. [2009]
Probability Models	Anomaly prediction and detection	Cohen et al. [2004], Zhang et al. [2005], Kandula et al. [2009], and Tan et al. [2010]
Statistical Process Control	Detection of performance transient and persistent anomalies and identification of anomalous correlation in database and enterprise systems	Trubin and Merritt [2004], Trubin [2005], Brey and Sironi [1990], Lee et al. [2012], Wang et al. [2013], and Bereznay and Permanente [2006]
Statistical Intervention Analysis	Bottleneck identification	Malkowski et al. [2007]

Internet applications, interventions are similar to phenomenal such as Internet *flash-crowds*, the *slashdot* effect, or a node failure.

3.2.2. Machine Learning Detection. Learning algorithms sift through massive metric space to identify patterns of interests or indistinct relationships [Rogers and Girolami 2011]. In performance studies, these patterns may be unexpected behaviours or symptoms of unplanned failures. Machine learning algorithms can be classified into two broad categories based on the nature of input and expected output of the algorithms [Alpaydin 2014].

Supervised Learning. Supervised learning algorithms require well-labeled datasets. Each data instance in a training dataset is assumed to belong to one of several classes (e.g., *normal* or *anomaly*). The goal is to build a generalized model that the captures the relationship between the feature set and each class during the training phase. These models are later used to classify new test instances during the testing phase. The need for well-labeled training data greatly limit the scope of their application for real-time use. They are, however, well suited for recognizing well-known anomalies. The use of supervised techniques in dynamic environments such as cloud data centers is hampered by the cost of retraining due to dynamic reconfiguration of application components and change in underlying execution environments. Supervised learning techniques do not easily lend themselves to frequent updates of training the dataset.

Unsupervised Learning. Unsupervised learning algorithms require no training data and no labeled data. The objective is to discover hidden patterns or regularities in the data, similar to density estimation in statistical. Unsupervised learning techniques cluster input data into classes based solely on their statistical properties. No assumption, however, is made of the distribution of the underlying data. For improved accuracy, it expected that normal data instances are more frequent in the dataset than abnormal instances. Techniques in this category are amenable to changes in the underlying system environment because no training is involved. And are particularly suitable for detecting unknown anomalies in cloud data centers where precise definition of anomaly characteristics may not always exist.

Semisupervised Learning. An emerging approach is to maximize the best of supervised and unsupervised learning. Semisupervised algorithms assume a small chunk of the dataset is labeled usually the normal class and the remaining unlabeled instances are anomalous. They often outperform their supervised and unsupervised counterparts as they leverage the presence of labeled data to identify inherent structure in the data. A similar approach is called the *weakly-supervised* or *bootstrapping* method. This method begins by training the classifier with a few training examples. When the classifiers finds positive test instances, it augments the original training data with the new instance and retrains the classifier. The performance of bootstrapping improves as the size of training data grows given false-positive detection is minimal. Bootstrapping is well suited for large-scale infrastructure where definitions of normality and abnormality evolve according to changing execution context.

The operation of a learning based PADBI system is often enhanced by two preprocessing tasks.

Dimensionality Reduction. To handle problems with many performance metrics, the metric space may be reduced by projecting the metrics to a new space where only the most relevant is preserved. Principal Component Analysis (PCA) is common

Table VI. Supervised PADBI Systems

Reference	Technique	Methodology
Tan and Adviser-Gu [2012] and Tan et al. [2010]	Bayesian classifier and Tree augmented networks (TAN)	Presents a method for predicting and classifying anomalies.
Fu [2011]	Decision trees and TAN	Presents a technique for reducing metric dimensions based on Mutual Information and PCA and identifying performance anomalies Tree-based classifier.
Jung et al. [2006]	Decision tree	Presents a decision-tree based automated approach for detecting performance bottlenecks.
Cohen et al. [2004]	Tree augmented Bayesian networks (TAN)	Proposes a method exploring TANs as a basis for detecting SLO violations and identifying sets of system metrics that caused the violation in multi-tier web applications.
Gu and Wang [2009]	Bayesian classification	A stream-based anomaly detection method is used to detect anomaly symptoms and infer their root-causes.
Parekh et al. [2006]	TAN, Bayesian networks, LogicBoost, C4.5 decision tree.	Explores the performance of various machine learning classifiers with regards to bottleneck detection in an enterprise applications.
Powers et al. [2005]	Bayesian classifiers, Auto-regressive models, Multivariate regression	Presents a comparative study of the performance of three machine learning and statistical methods to predict the number of performance SLA violations.

method for doing this. PCA takes k correlated metrics as input and reduces them to $m \leq k$ nondependent metrics. These m metrics can be interpreted as linear combinations of the original set [Guan et al. 2012; Fu 2011]. Other methods for dimension reduction include factor analysis, independent component analysis, and nonlinear PCA [Fodor 2002].

Similarity Identification. Metrics with high similarity affects the efficiency of learning algorithms such as clustering. A common method of evaluating similarities between features is based on the *Mutual Information* algorithm from the domain of Information Theory [Steuer et al. 2002; Battiti 1994].

Unlike statistical detection, learning techniques do not make assumptions about the underlying distribution of data. We identify a few references of each type of learning in Tables VI, VII, and VIII.

We further describe commonly used learning techniques in literature in Sections 3.2.2.1 through 3.2.2.4.

3.2.2.1. Classification-based Techniques. Classification-based learning algorithms are special cases of supervised learning. The objective is to determine if data instances in a given feature space belongs to one class or multiple classes. During a training phase, the algorithm identify classes and learn a model that associate each class label with the characteristics of features present in the data. The testing phase use these models to classify new data samples. Ruled-based detection systems are a specific example of how classification learning can be used in detecting anomalous behaviours. Common classification techniques include *Decision Trees*, *Support Vector Machines*, *Artificial Neural Networks*, and *Bayesian Networks* [Kotsiantis 2007].

Rule-based techniques. The goal is to learn as many rules that captures normal behaviours of a system as possible. First they discover rules from the training data

Table VII. Unsupervised PADBI Systems

Reference	Technique	Methodology
Wang et al. [2012, 2014]	Local Outlier Factor (LOF)	Proposes an online anomaly detection approach for web applications present an incremental clustering algorithm for training workload patterns online, and employ LOF in the recognized workload pattern to detect anomalies.
Dean et al. [2012]	Self-Organizing Maps (SOM)	Presents an anomaly detection mechanism in IaaS Cloud using SOMs to learn emergent system behaviour and predict unknown anomalies.
Guan et al. [2012]	Bayesian ensemble models	Proposes an hybrid learning approach by characterizing normal execution states of the system as an ensemble of unsupervised Bayesian models and uses decision tree to predict and detect system failures in a Cloud environment.
Huang et al. [2013]	Local Outlier Factor (LOF)	Presents an adaptive method extending the Local Outlier Factor algorithm for detecting both contextual and unknown anomalies in a Cloud system.
Yu and Lan [2013]	Nonparametric clustering	Proposes a decentralized approach for detecting anomalies in Hadoop clusters based on Hierarchical Grouping and majority voting.

Table VIII. Semisupervised PADBI Systems

Reference	Technique	Methodology
Lan et al. [2010]	Principal and Independent Component Analysis	Presents an automated anomaly detection mechanisms for identifying system nodes whose behaviours are deviating from others in a cloud data center.
Pannu et al. [2012]	Classification, Clustering, Support Vector Machines	Presents a self-evolving mechanism for predicting and detecting of system failures in Cloud systems.
Smith et al. [2010]	Bayesian Networks, Principal Component Analysis, Clustering	Presents an autonomic mechanism for anomaly detection in a compute Cloud system using PCA and Bayesian models for feature extraction and Expectation Maximization clustering algorithm for anomaly detection.
Bhaduri et al. [2011]	K-Nearest Neighbours	Proposes an automated failure detection system employing distance-based anomaly rules to identify faulty machines in a cluster.
Guan and Fu [2013b] and Guan et al. [2013]	Wavelets	Presents a method analyzing performance metrics in both time and frequency domains in order to identify anomalous behaviors in a Cloud environment.
Fu et al. [2012]	Support Vector Machines (SVM)	Proposes an hybrid self-evolving anomaly detection framework using one-class and two-class SVM.

using *Decision Trees*, *Association Rules*, *C4.5* classification. During the testing phase, for each test instance, the best rule that captures the instance is used to compute an anomaly score for designating the test instances as anomalous or normal. A rule has an associated confidence score proportional to the ratio between the number of correct classification by the rule and total number of cases covered by the rule. The anomaly score is computed as the inverse of the confidence score associated with a given rule [Chandola et al. 2009].

The complexity of classification techniques depends on the algorithms used. Training decision trees is often faster than training techniques such as SVM that involves quadratic optimization. The testing phase is also faster. Classification methods rely heavily on accurately labeled data, and also produces class labels which may not be useful in cases where an associated score is required.

3.2.2.2. Neighbour-based Techniques. Unlike classification-based approaches, neighbour-based techniques are unsupervised learning systems that evaluates data instances based on its local neighbourhood. The assumption is that normal data usually occur in dense neighbourhoods while abnormal data occur far from their closest neighbour [Chandola et al. 2009]. It is also required that a distance or similarity measure is estimated between two data instances depending on the data type. Different methods exist for calculating similarity measures such as Euclidean Distance for continuous data and Mutual Information for categorical data. A popular neighbourhood method is the *k*th-Nearest Neighbour which estimates the distance of a given instance to its nearest neighbours and evaluate the distance against a predefined domain specific threshold [Liao and Vemuri 2002; Lazarevic et al. 2003]. The Local Outlier Factor (LOF) algorithm is another neighbour-based technique that detect anomalous instances by estimating the density of each instance. Instances in low-density neighbourhoods are classified as anomalous [Wang et al. 2012]. Basic neighbour-based and LOF methods has a time complexity of $O(N^2)$. Its testing phase is computationally intensive because distance score of a test instance to others is required. It is also difficult to create distance measures for complex data (e.g., spatial and streaming data).

3.2.2.3. Clustering-based Techniques. Clustering is another type of unsupervised learning that groups similar data instances into clusters according to hidden relationships between instances in a cluster [Berkhin 2006]. The goal is to find clusters of similar data points such that each cluster is well separated. Detection of anomalous instances can be based on the density of the clusters (e.g., dense or sparse) or distance of instances from the closest centroid in the cluster [Chandola et al. 2009]. The Euclidean distance, Mahalanobis distance, and Cosine similarity are example of distance measures for such cases. Examples of clustering algorithms include the K-means clustering, Expectation Maximization (EM), and Self-Organizing Maps (SOM) [Hodge and Austin 2004]. Time complexity of clustering depends on the algorithm in use. Testing phase is faster since test instances are compared with only a few cluster.

3.2.2.4. Information Theoretic Techniques. Information theory provides many measures for estimating the degree of dispersal or concentration of the information content of a data set [Wang et al. 2010]. The primary assumption of these methods is that anomalies induce irregularities in the information content of a given dataset [Chandola et al. 2009]. Also they are very generic in nature with no need for parameterization [Wagner and Plattner 2005]. The Entropy information measure or Shannon-Wiener Index [Shannon 2001] estimates the degree of uncertainty in a given dataset. Given a random variable X, its entropy is computed as $H(X) = -\sum_{i=1}^{n} P(x_i)\log(P(x_i))$, where $P(x)$ is the probability distribution of X. The entropy $H(X)$ lies in the range

$[0, \log(n)]$. Higher entropy values indicate more randomness in the data and may be more anomalous than data with lower $H(X)$ values [Navaz et al. 2013]. The degree of randomness between two random variables with probability distributions $P(x)$ and $Q(x)$ can be estimated by their Relative Entropy, $H(Q|P) = \sum Q(x) \log \frac{Q(x)}{P(x)}$. An application of this is to compare the entropy values of two different windows of observation of a metric for detecting changes. The smaller the relative entropy the better. A $H(Q||P)$ value of 0 indicates that the probability distributions $P(X)$ and $Q(X)$ exhibit the same randomness [Lee and Xiang 2001]. Entropy-based methods have been applied to study malicious behaviours in network traffic in Wagner and Plattner [2005] and Lee and Xiang [2001]. Wang et al. [2009, 2010] present entropy-based methodologies for detecting anomalies in a cloud computing environment by analyzing metric distributions. Entropy generally provide more fine-grained insights of the data than traditional classification methods [Nychis et al. 2008] and suitable for online unsupervised detection of unknown anomalies [Wang et al. 2010] since no assumptions of underlying distribution is made.

4. RESEARCH TRENDS

Before the 2000s, contributions focused primarily on the detection of coarse-grained performance issues such as identifying hardware, software bottlenecks in the operating systems [Mahapatra and Venkatrao 1999; Breese and Blake 1995], networks [Melander et al. 2000], and client-server applications [Neilson et al. 1995].

Due to the emergence of the Internet, the early 2000s witnessed a slow trend towards web and distributed applications hosted in dedicated environments. Chen et al. [2002], Aguilera et al. [2003], and Barham et al. [2003] proposed techniques for uncovering performance failures and anomalies with regards to web and distributed systems. By the mid to late 2000s, efforts concentrated on building improved detection mechanisms targeting enterprise applications running in shared-hosting environments, grid, and large-scale infrastructures. This period witnessed the development of analytical approaches and tools such as transaction mix models [Kelly 2005b], queuing-theoretic models [Kelly 2005a], signature models [Mi et al. 2008b; Cherkasova et al. 2008], and statistical techniques [Malkowski et al. 2007; Cherkasova et al. 2009]. While efforts such as in Jung et al. [2006] and Malkowski et al. [2009] propose an experimental approach, Chung et al. [2008] and Agarwala et al. [2007] demonstrate the potential of analyzing the flow of messages across distributed components as a suitable method for detecting performance abnormalities.

From the late 2000s until now, the research contributions have been largely consolidated on achieving dependability [Guan and Fu 2013b; Lee et al. 2012] predictable performance [Tan et al. 2010], root-cause identification [Magalhaes and Moura Silva 2011; Bhaduri et al. 2011; Yu and Lan 2013] and meeting performance guarantees [Kang et al. 2012; Lan et al. 2010] in cloud computing applications and systems. Similarly, there are systems tailored to detecting and resolving workload related anomalies [Wang et al. 2012, 2014]. Perhaps due to scale and the special requirements imposed by the cloud, advanced machine learning techniques have found extensive use in bottleneck and anomaly detection research [Dean et al. 2012, 2014; Huang et al. 2013; Sharma et al. 2013]. Even though existing research contribution is dominated by reactive solutions, there is increasing shift toward proactive approach. Predictive anomaly and bottleneck detection offers better system reliability by raising in advance, just-in-time alerts and detecting potential bottlenecks before a performance issue occur. Examples of such approach can be found in Guan et al. [2011] and Tan and Adviser-Gu [2012].

Following the trends, we observe that cloud computing systems and applications will continue to attract the attention of performance anomaly detection and bottleneck identification research. Characteristics of the cloud systems such as heterogeneity of resources and application services, variable load, and performance variation complicate the problem of detecting performance issue [Gong et al. 2010]. We describe these challenges in detail in Section 5.

5. PADBI SYSTEMS IN THE CLOUD: SPECIFIC REQUIREMENTS

Cloud computing enables computing resources to be provisioned on demand as an utility over the Internet and dynamically scale in response to unpredictable demands and application workloads. A cloud infrastructure is typically characterized by a pool of heterogeneous hardware and software resources that are shared by many application services with disparate performance objectives [Zhang et al. 2010; Jennings and Stadler 2014]. The resulting resource contention and performance interference caused by resource sharing have significant impact on the performance of cloud services and systems [Sharma et al. 2013; Wang et al. 2010].

Inherent characteristics of the cloud such as the heterogeneity of resource types and their interdependencies; the variability and unpredictability of load; and the complex architecture of cloud services; make the task of detecting and resolving performance problems more difficult. To meet stringent performance objectives and to achieve predictable performance, PADBI systems must take into consideration specific cloud requirements as described in the following text.

(1) *Scale.* Medium- to large-scale cloud infrastructures run up to thousands of applications on limited computing resources. It is daunting to keep track of the execution status of such huge applications base [Tan et al. 2012; Dean et al. 2012]. Considering that these applications are composed of multiple service components and the complex topology of the infrastructure, the potential metric space is huge. Wang et al. [2010] estimates this to the Exa scale. That is up to 10^{18} metrics to monitor and process in real time! This require PADBI systems to be lightweight with negligible performance and storage overhead. Also, they must be able to operate in an online fashion in order to keep up with the time varying nature of the cloud.
(2) *Multitenancy.* Multitenancy enables different applications (deployed in virtual machines (VMs)) to be colocated on the same physical server. These VMs concurrently share and compete for virtualized resources (such as CPU and memory) and non virtualized resources (such as network and caches). Such a tight execution environment has been shown to account for 40% in performance degradation in some applications [Sharma et al. 2013]. This makes it essential for PADBI techniques to be aware of prevailing execution contexts.
(3) *Complex Application Architecture.* The cloud run an heterogeneous mix of applications with time-varying workload patterns, ranging from long-running MapReduce jobs and HPC scientific workflows; to interactive web-based social media platforms, e-commerce, and media streaming applications [Wang et al. 2010]. Also, many of these applications share temporal dependency such as two applications having similar workload behaviours. Moreover, services in IaaS clouds come in black-boxes with limited visibility by the cloud infrastructure provider. This limits the extent to which performance degradation issues can be diagnosed and resolved [Dean et al. 2012; Tan et al. 2012].
(4) *Dynamic Resource Management.* Due to the continuous flow of load in and out of the cloud, resource management tasks such as dynamic reconfiguration, consolidation and migration constantly change the operational context in which applications runs

[Tan et al. 2012; Wang et al. 2010]. This leads to a higher frequency of anomalies. Faulty VM reconfigurations, and spontaneous live migrations have been observed to impact performance by up to 30% and 10%, respectively [Sharma et al. 2013]. In such environments, it is nearly difficult to determine what performance behaviour is normal and which is not [Dean et al. 2012].

(5) *Autonomic Management.* Today's data centers are powered by highly automated mechanisms. Autonomic resource managers dynamically provision resources based on adaptive system policies to meet expected quality of service (QoS) and achieve optimal resource utilization levels [Buyya et al. 2012; Hasan et al. 2012]. Delayed detection and manual resolutions do not fit the cloud model, as they can cause prolonged performance violations with huge financial penalty and failure [Dean et al. 2012; Tan et al. 2012]. Therefore, PADBI systems for the cloud are must be dynamic and proactive in nature [Sharma et al. 2013; Wang et al. 2010].

6. DISCUSSIONS AND FUTURE DIRECTIONS

The motivation for detecting unexpected performance behaviours and their root-causes is due to the significant impact they have on smooth operation of systems, the criticality of information they bear, and the costly penalties due to loss of dissatisfied users. The choice of detection is influenced not only by the characteristics of the anomalies and bottlenecks of interest but also by the nature of data and system under test.

PADBI systems based on statistical methods are only as correct as the correctness of the data, the assumption of its distribution and the fitness of the analysis. It is very important to collect the right data and quantity. Care must be taken to balance the proportion of normal samples to anomaly samples in the dataset to avoid the "needle in a haystack" [6] problem. Though parametric techniques assume known data distribution and best at identifying well-known anomalies, nonparametric methods are resistant to high variation in the data without knowledge of data distribution.

Machine learning solutions can quickly sift through a massive metric space to identify patterns of interests or indistinct relationships. Learning techniques expect that normal data instances are more frequent in the data; otherwise, they suffer from high false detection. While most classification, clustering, and statistical techniques have expensive training phases, they provide fast testing with high false-positive detection when unknown anomalous data is frequent. On the other hand, neighbour-based learning methods require no training phase and are highly suitable for real-time detection. However, they are computationally expensive.

Further advancement in hybrid solutions holds great potential for today's system such as proposed in Fu et al. [2012]. Rigid assumptions (regarding distribution and density of performance data) imposed by statistical techniques do not always work in dynamic environments. In addition, unsupervised algorithms are known to perform poorly in cases where anomalies occur more frequently in the test data than normal. When deciding the choice of methods to use in a given case, it is important to consider the tradeoff between online and offline detection as well as the cost incurred when there is a requirement for frequent model updates. Today's systems are dynamic with constant changing execution contexts, application composition, and configurations. It is also expected that anomaly detection and bottleneck identification mechanisms are able to *adapt* as well. Methods that require extensive training phase is inadequate in this case. Focus then must be on techniques that support online updates of model parameters and variables.

[6]A situation where it is nearly impossible to detect anomalous instances in the dataset because only a few anomalous instances exist in the training data.

Tables IX and X of Appendix A summarize major references used in this work based on the essential characteristics of the PADBI problem. Furthermore, we have identified a few promising directions and open challenges within the scope of the problem and briefly outline them in the following text:

(1) *Multilevel bottleneck detection.* Current efforts must extend toward the detection of performance bottlenecks at different levels considering the complexity of today's infrastructure and application. For instance, it should be possible to identify bottlenecks from a set of top-level application service components and further down through the virtualization layer to system resource bottlenecks. Similarly, anomaly detection should be viewed from three perspectives: workload, resource demand and performance.
(2) *Taxonomy of performance bottlenecks and anomalies.* A taxonomy of performance issues under various operational condition (e.g., workload, platform) and manifestation will be highly essential for industry and academia. The challenge here is that these behaviours are inherently intrinsic to the applications and their manifestations vary from one application to another. However, we believe little steps can be made toward this especially for common performance anomalies and bottlenecks. A similar direction is documented in Pertet and Narasimhan [2005].
(3) *Open performance datasets.* Their lack of open performance datasets hinders the pace of research in this area because such data are often considered highly sensitive or classified. Google Cluster [Reiss et al. 2011] trace serves a similar purpose. However, the Google data is an old 29-day trace of a 12,000-machine cluster covering jobs, tasks, resource usage, and machine events measurement from 2011. Similarly, the Yahoo Webscope [Yahoo! 2014] project provides system measurements of the infrastructure running its cloud serving benchmark system [Cooper et al. 2010]. However, the data covers only resource usage across system components over a mere 30-minute period. Due to sensitivity, these datasets do not contain performance metrics such as throughput and latency. Similar lack of dataset for failure detection research is acknowledged by Schroeder et al. [2010].
(4) *Anomaly-resistant resource allocation.* The autonomic nature of modern IT infrastructures demands tight integration of proactive anomaly detection mechanisms with autonomic resource managers. Alerting administrators of an anomaly delays the detection and resolutions of performance problems. This semiautomated approach does not fit today's model of system management, where prolonged performance violations may induce significant unplanned downtimes.
(5) *Context-aware detection.* Frequent performance variations exhibited by cloud applications have been attributed to the changing execution context of the underlying environment. This is often due to frequent workload variation and dynamic resource reconfiguration. The challenge is identifying and characterizing execution contexts as they evolve over time. Context-aware solutions capable of achieving this in addition to adapting to nonstationary cloud behaviours will greatly improve application performance. Tan et al. [2010] and Tan and Gu [2010], and Sharma et al. [2013] present interesting directions in this case.
(6) *Distributed detection.* A huge chunk of current research focus is on centralized detection. Modern enterprise systems are inherently distributed with components spanning multiple physical domains (servers or data centers). Often times the collection of data across such domains is impractical or difficult due to potential system overheads and proprietary and privacy regulations. This implication calls for a decentralized approach that fits naturally with such systems. A theoretical

attempt is presented in Lazarevic et al. [2009], while a similar case study for failure detection is studied in Bhaduri et al. [2011].

7. CONCLUDING REMARKS

We present a review of the performance anomaly detection and bottleneck identification problem and identify relevant research questions, challenges, contributions, trends and open issues. For clarity, we highlight different types of commonly observed performance anomalies and bottlenecks in computing systems. Existing PADBI systems operate based on one or more detection strategies and methods. Statistical and machine learning are the two predominant methods in literature. We have highlighted major classes of techniques in both methods along with interesting references. The choice of strategies and techniques is largely influenced by the goal of the system and the core elements of the problem such as the nature of the system or application, the performance data, and the extent to which the system can be observed. Based on trends, the problem of detecting performance issues and their root-causes will continue to attract research attention, especially in cloud services. We also highlighted specific requirements for effective anomaly and bottleneck detection in cloud computing infrastructures. However, the problem of multilevel bottleneck detection, distributed detection, and accessible performance datasets still remain open research issues.

A. APPENDIX A

B. GENERAL OVERVIEW OF PADBI SYSTEMS

Table IX. Overview of PADBI Systems

Reference	Goal	System	Observability	Strategy	Method	Techniques
Chen et al. [2002]	PBI	Distributed, Web-based, Component bottlenecks	White-box, Source Tracing	Flow & Dependency	Hybrid	Clustering, Correlation
Cohen et al. [2004]	PADBI	Multi-tier, Web-based, System metrics	Black-box, Profiling	Observational	Machine Learning	Tree Augmented Bayesian Networks
Kelly [2005a]	PAD	Distributed, Web-based, Application & System metrics	Gray-box, Profiling	Observational	Statistical	Regression, Transaction mix Model
Cohen et al. [2005]	PAD	Distributed, Enterprise, Application & System metrics	Black-box, Profiling	Signature-based	Machine Learning	Clustering, Tree-Augmented Naive Bayes Models
Jung et al. [2006]	PADBI	Multi-tier, Application & System metrics	Black-box, Profiling	Observational	Machine Learning	C4.5 Decision Tree
Malkowski et al. [2007]	PBI	Web-based, System metrics	Gray-box, Profiling	Knowledge-based	Statistical	SIA

(Continued)

Table IX. Continued

Reference	Goal	System	Observability	Strategy	Method	Techniques
Agarwala et al. [2007]	PADBI	Distributed, Multi-tier, Application & System metrics	White-box	Flow & Dependency	Statistical	Correlation
Zhang et al. [2007b]	PAD	Multi-tier, System & Application metrics	Gray, Profiling	Observational	Statistical	Regression, TM models, Queuing model
Gunter et al. [2007]	PAD	Grid, System metrics	Black-box, Logging	Observational	Statistical	MSD, CDF, EWMA
Yang et al. [2007]	PAD	Grid, System resource metrics	Black-box, Profiling	Observation, Flow & Dependency	Statistical, Signal Processing	Extended Window Averaging, Regression
Chung et al. [2008]	PBI	HPC, Resource bottlenecks	Gray, Profiling	Knowledge-based	-	Inference Engine
Cherkasova [et al. 2008]	PAD	Web-based, Multi-tier, Application & System metrics	Gray, Profiling	Signature-based	Statistical	Regression, Transaction mix Model
Bodík et al. [2008]	PAD	Distributed Systems, System metrics	Black-box, Profiling	Signature-based	Machine Learning	Logistic Regression with L1 Regularization
Malkowski et al. [2009]	PADBI	Multi-tier, Application & System metrics	Gray	Observational	Statistical, Signal Processing	Kernel Density Estimation, Adaptive Filtering
Wang et al. [2009]	PBI	Multi-tier	Gray	Observational	-	Heuristics
Kandula et al. [2009]	PADBI	Enterprise Systems	Gray-box	Knowledge-based, Flow & Dependency	Statistical Learning	Probability Models, Inference Engine
Ben-Yehuda et al. [2009]	PBI	Virtualized, Component & Resource bottlenecks	Black-box, Profiling	Flow & Dependency	Statistical, Queueing Theory	Percentile Testing, Little's Law
Iqbal et al. [2010]	PBI	Cloud, Multi-tier, Resource bottlenecks	Black-box, Profiling	Observational	-	-
Bodik et al. [2010]	PAD	Distributed Systems, Cloud, System metrics	Black-box, Profiling	Signature-based	Statistical and Machine Learning	Quantile Summarization, Logistic Regression with L1 Regularization

Table X. Overview of PADBI Systems (cont.)

Reference	Goal	System	Observability	Strategy	Method	Techniques
Lan et al. [2010]	PAD	Cloud, Host/Node bottlenecks	Black-box, Profiling	Observational	Machine Learning	Principal and Independent Component Analysis
Magalhaes and Moura Silva [2011]	PADBI	Web-based, Application metrics	White-box, Request tracing	Flow & Dependency	Statistical	Correlation Analysis, ANOVA
Fu [2011]	PAD	Cloud, System metrics	Black-box, Profiling	Knowledge-based	Machine Learning	Mutual Information, PCA, Semi-supervised Decision-tree
Sambasivan et al. [2011]	PAD	Distributed, Storage, Network request flows	Gray, Tracing	Flow & Dependency	Hybrid	C4.5, Regression Tree
Tan et al. [2012]	PAD	Cloud, Web-based, System metrics	Black-box, Profiling	Observational	Hybrid	Markov-model, Tree Augmented Bayes
Pannu et al. [2012]	PAD	Cloud, System metrics	Black-box, Profiling	Knowledge-driven	Machine Learning	Supervised, One-class SVM
Lee et al. [2012]	PADBI	Distributed, Storage, Application metrics	Black-box, Profiling	Knowledge-base	Statistical	SPC
Kang et al. [2012]	PADBI	Cloud, Application & System metrics	Black-box, Profiling	Observational	Hybrid	Regression (LAR), Clustering
Dean et al. [2012]	PADBI	Cloud, System metrics	Black-box, Profiling	Observational	Machine Learning	Self-Organizing Maps
Wang et al. [2012]	PAD	Web-based, System metrics	Gray-box, Profiling	Observational	Machine Learning	Clustering, LOF

(Continued)

Table X. Continued

Reference	Goal	System	Observability	Strategy	Method	Techniques
Nguyen et al. [2013]	PADBI	Cloud	Black-box	Flow & Dependency	Statistical, Signal Processing	CUSUM, FFT Filtering
Guan and Fu [2013b]	PAD	Cloud, System metrics	Black-box, Profiling	Observational	Machine Learning	Wavelet, Sliding Window
Wang et al. [2013b]	PBI	Multitier, Component & Resource bottlenecks	Blackbox	Observation	-	Fine-grained Load, Throughput Analysis
Xiong et al. [2013]	PAD	Distributed, Resource metrics	Blackbox, Profiling	Observation	Statistical	Correlation Analysis
Sharma et al. [2013]	PAD	Cloud, Web-based, System metrics	Black-box, Profiling	Observational	Machine Learning	Hidden Markov Model, k-Nearest Neighbour, K-means Clustering
Huang et al. [2013]	PAD	Cloud, System metrics	Black-box, Profiling	Knowledge-based	Machine Learning	LOF
Yu and Lan [2013]	PAD	Distributed, Hadoop Clusters, Component and Host bottlenecks	Black-box, Profiling	Observational	Machine Learning	Non-parametric Clustering
Wang et al. [2014]	PAD	Web-based, System metrics	Gray-box, Profiling	Observation	Machine Learning	Clustering, LOF
Dean et al. [2014]	PADBI	Cloud, System metrics	Black-box, System-call tracing	Observation	Statistical	Tukey Limits

REFERENCES

Sandip Agarwala, Fernando Alegre, Karsten Schwan, and Jegannathan Mehalingham. 2007. E2EProf: Automated end-to-end performance management for enterprise systems. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. IEEE, 749–758.

Marcos K. Aguilera, Jeffrey C. Mogul, Janet L. Wiener, Patrick Reynolds, and Athicha Muthitacharoen. 2003. Performance debugging for distributed systems of black boxes. *ACM SIGOPS Operating Systems Review* 37, 74–89.

E. Alpaydin. 2014. *Introduction to Machine Learning*. MIT Press.

Paul Barham, Rebecca Isaacs, Richard Mortier, and Dushyanth Narayanan. 2003. Magpie: Online modelling and performance-aware systems. In *Proceedings of the 9th Workshop on Hot Topics in Operating Systems (HotOS IX)*. 85–90.

Roberto Battiti. 1994. Using mutual information for selecting features in supervised neural net learning. *IEEE Transactions on Neural Networks* 5, 4, 537–550.

Muli Ben-Yehuda, David Breitgand, Michael Factor, Hillel Kolodner, Valentin Kravtsov, and Dan Pelleg. 2009. NAP: A building block for remediating performance bottlenecks via black box network analysis. In *Proceedings of the 6th International Conference on Autonomic Computing*. ACM, 179–188.

Frank M. Bereznay and Kaiser Permanente. 2006. Did something change? using statistical techniques to interpret service and resource metrics. In *Proceedings of the International CMG Conference*. 229–242.

Pavel Berkhin. 2006. A survey of clustering data mining techniques. In *Grouping Multidimensional Data*. Springer, 25–71.

Kanishka Bhaduri, Kamalika Das, and Bryan L. Matthews. 2011. Detecting abnormal machine characteristics in cloud infrastructures. In *Proceedings of the IEEE 11th International Conference on Data Mining Workshops (ICDMW'11)*. IEEE, 137–144.

Walter Binder, Jarle Hulaas, and Philippe Moret. 2007. Advanced java bytecode instrumentation. In *Proceedings of the 5th International Symposium on Principles and Practice of Programming in Java*. ACM, 135–144.

Peter Bodík, Moises Goldszmidt, and Armando Fox. 2008. HiLighter: Automatically building robust signatures of performance behavior for small-and large-scale systems. In *SysML*. USENIX Association.

Peter Bodik, Moises Goldszmidt, Armando Fox, Dawn B. Woodard, and Hans Andersen. 2010. Fingerprinting the datacenter: Automated classification of performance crises. In *Proceedings of the 5th European Conference on Computer Systems*. ACM, 111–124.

George E. P. Box and George C. Tiao. 1975. Intervention analysis with applications to economic and environmental problems. *J. Amer. Statist. Assoc.* 70, 349, 70–79.

John S. Breese and Russ Blake. 1995. Automating computer bottleneck detection with belief nets. In *Proceedings of the 11th Conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann, 36–45.

Jack Brey and Rick Sironi. 1990. Managing at the knee of the curve (The use of SPC in managing a data center). In *Proceedings of the International CMG Conference*. 895–901.

Shaun Burke. 2001. Missing values, outliers, robust statistics & non-parametric methods. *LC-GC Europe Online Supplement, Statistics & Data Analysis* 2, 19–24.

Rajkumar Buyya, Rodrigo N. Calheiros, and Xiaorong Li. 2012. Autonomic cloud computing: Open challenges and architectural elements. In *Proceedings of the 3rd International Conference on Emerging Applications of Information Technology (EAIT'12)*. IEEE, 3–10.

Jeffrey P. Buzen and Annie W. Shum. 1995. Masf-multivariate adaptive statistical filtering. In *Proceedings of the International CMG Conference*. 1–10.

Giuliano Casale, Amir Kalbasi, Diwakar Krishnamurthy, and Jerry Rolia. 2009. Automatic stress testing of multi-tier systems by dynamic bottleneck switch generation. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*. Springer-Verlag New York, 20.

Giuliano Casale, Ningfang Mi, Ludmila Cherkasova, and Evgenia Smirni. 2012. Dealing with burstiness in multi-tier applications: Models and their parameterization. *IEEE Transactions on Software Engineering* 38, 5, 1040–1053.

Varun Chandola, Arindam Banerjee, and Vipin Kumar. 2009. Anomaly detection: A survey. *ACM Computing Surveys (CSUR)* 41, 3, 15.

Mike Y. Chen, Emre Kiciman, Eugene Fratkin, Armando Fox, and Eric Brewer. 2002. Pinpoint: Problem determination in large, dynamic internet services. In *Proceedings of International Conference on Dependable Systems and Networks*. IEEE, 595–604.

Ludmila Cherkasova, Kivanc Ozonat, Ningfang Mi, Julie Symons, and Evgenia Smirni. 2008. Anomaly? application change? or workload change? Towards automated detection of application performance anomaly

and change. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks with FTCS and DCC*. IEEE, 452–461.

Ludmila Cherkasova, Kivanc Ozonat, Ningfang Mi, Julie Symons, and Evgenia Smirni. 2009. Automated anomaly detection and performance modeling of enterprise applications. *ACM Transactions on Computer Systems (TOCS)* 27, 3, 6.

I-Hsin Chung, Guojing Cong, David Klepacki, Simone Sbaraglia, Seetharami Seelam, and Hui-Fang Wen. 2008. A framework for automated performance bottleneck detection. In *Proceedings of the IEEE International Symposium on Parallel and Distributed Processing (IPDPS'08)*. IEEE, 1–7.

Ira Cohen, Jeffrey S. Chase, Moises Goldszmidt, Terence Kelly, and Julie Symons. 2004. Correlating instrumentation data to system states: A building block for automated diagnosis and control. In *OSDI*, Vol. 4. 16–16.

Ira Cohen, Steve Zhang, Moises Goldszmidt, Julie Symons, Terence Kelly, and Armando Fox. 2005. Capturing, indexing, clustering, and retrieving system history. In *ACM SIGOPS Operating Systems Review*, Vol. 39. ACM, 105–118.

Brian F. Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. 2010. Benchmarking cloud serving systems with YCSB. In *Proceedings of the 1st ACM Symposium on Cloud Computing*. ACM, 143–154.

Marc Courtois and Murray Woodside. 2000. Using regression splines for software performance analysis. In *Proceedings of the 2nd International Workshop on Software and Performance*. ACM, 105–114.

Kaustav Das. 2009. Detecting patterns of anomalies. Technical Report CMU-ML-09-101. PhD thesis. Carnegie Mellon University, Department of Machine Learning.

Daniel Joseph Dean, Hiep Nguyen, and Xiaohui Gu. 2012. Ubl: Unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems. In *Proceedings of the 9th International Conference on Autonomic Computing*. ACM, 191–200.

Daniel J. Dean, Hiep Nguyen, Peipei Wang, and Xiaohui Gu. 2014. PerfCompass: Toward runtime performance anomaly fault localization for infrastructure-as-a-service clouds. In *Proceedings of the 6th USENIX Conference on Hot Topics in Cloud Computing*. USENIX Association, 16–16.

Anh Vu Do, Junliang Chen, Chen Wang, Young Choon Lee, Albert Y. Zomaya, and Bing Bing Zhou. 2011. Profiling applications for virtual machine placement in clouds. In *Proceedings of the IEEE International Conference on Cloud Computing (CLOUD'11)*. IEEE, 660–667.

Evolven. 2011. Downtime, Outages and Failures—Understanding Their True Costs. Retrieved March 11, 2015 from http://www.evolven.com/blog/downtime-outages-and-failures-understanding-their-true-costs.html.

Imola K. Fodor. 2002. A survey of dimension reduction techniques. Technical Report UCRL-ID-148494. Lawrence Livermore National Laboratory.

Song Fu. 2011. Performance metric selection for autonomic anomaly detection on cloud computing systems. In *Proceedings of the Global Telecommunications Conference (GLOBECOM'11)*. IEEE, 1–5.

Song Fu, Jianguo Liu, and Husanbir Pannu. 2012. A Hybrid anomaly detection framework in cloud computing using one-class and two-class support vector machines. In *Advanced Data Mining and Applications*. Springer, 726–738.

Alessio Gambi and Giovanni Toffetti. 2012. Modeling cloud performance with kriging. In *Proceedings of the 2012 International Conference on Software Engineering*. IEEE Press, 1439–1440.

Chunye Gong, Jie Liu, Qiang Zhang, Haitao Chen, and Zhenghu Gong. 2010. The characteristics of cloud computing. In *Proceedings of the 39th International Conference on Parallel Processing Workshops (ICPPW'10)*. IEEE, 275–279.

Brendan Gregg. 2013. *Systems Performance: Enterprise and the Cloud*. Pearson Education.

Frank E. Grubbs. 1969. Procedures for detecting outlying observations in samples. *Technometrics* 11, 1, 1–21.

Xiaohui Gu and Haixun Wang. 2009. Online anomaly prediction for robust cluster systems. In *Proceedings of the IEEE 25th International Conference on Data Engineering (ICDE'09)*. IEEE, 1000–1011.

Qiang Guan and Song Fu. 2013a. Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures. In *Proceedings of the IEEE 32nd International Symposium on Reliable Distributed Systems (SRDS'13)*. IEEE, 205–214.

Qiang Guan and Song Fu. 2013b. Wavelet-based multi-scale anomaly identification in cloud computing systems. In *Proceedings of the Global Communications Conference (GLOBECOM'13)*. IEEE, 1379–1384.

Qiang Guan, Song Fu, Nathan DeBardeleben, and Sean Blanchard. 2013. Exploring time and frequency domains for accurate and automated anomaly detection in cloud computing systems. In *Proceedings of the IEEE 19th Pacific Rim International Symposium on Dependable Computing (PRDC'13)*. IEEE, 196–205.

Qiang Guan, Ziming Zhang, and Song Fu. 2011. Proactive failure management by integrated unsupervised and semi-supervised learning for dependable cloud systems. In *Proceedings of the 6th International Conference on Availability, Reliability and Security (ARES'11)*. IEEE, 83–90.

Qiang Guan, Ziming Zhang, and Song Fu. 2012. Ensemble of bayesian predictors and decision trees for proactive failure management in cloud computing systems. *Journal of Communications* 7, 1, 52–61.

Dan Gunter, Brian L. Tierney, Aaron Brown, Martin Swany, John Bresnahan, and Jennifer M. Schopf. 2007. Log summarization and anomaly detection for troubleshooting distributed systems. In *Proceedings of the 8th IEEE/ACM International Conference on Grid Computing*. IEEE, 226–234.

Neil J. Gunther. 2004. Benchmarking blunders and things that go bump in the night. *CoRR*. http://arxiv.org/abs/cs.PF/0404043

Neil J. Gunther. 2011. *Analyzing Computer System Performance with Perl:: PDQ*. Springer.

Masum Z. Hasan, Edgar Magana, Alexander Clemm, Lew Tucker, and Sree Lakshmi D. Gudreddi. 2012. Integrated and autonomic cloud resource scaling. In *Proceedings of the Network Operations and Management Symposium (NOMS'12)*. IEEE, 1327–1334.

Victoria J. Hodge and Jim Austin. 2004. A survey of outlier detection methodologies. *Artificial Intelligence Review* 22, 2, 85–126.

Cheng Huang. 2011. Public DNS System and Global Traffic Management. Retrieved April 15, 2014 from http://research.microsoft.com/en-us/um/people/chengh/slides/pubdns11.pptx.pdf.

Su-Yun Huang, Mei-Hsien Lee, and Chuhsing Kate Hsiao. 2006. Kernel canonical correlation analysis and its applications to nonlinear measures of association and test of independence. Institute of Statistical Science: Academia Sinica, Taiwan.

Tian Huang, Yan Zhu, Qiannan Zhang, Yongxin Zhu, Dongyang Wang, Meikang Qiu, and Lei Liu. 2013. An LOF-based adaptive anomaly detection scheme for cloud computing. In *Proceedings of the IEEE 37th Annual Computer Software and Applications Conference Workshops (COMPSACW'13)*. IEEE, 206–211.

Waheed Iqbal, Matthew N. Dailey, David Carrera, and Paul Janecek. 2010. SLA-driven automatic bottleneck detection and resolution for read intensive multi-tier applications hosted on a cloud. In *Advances in Grid and Pervasive Computing*. Springer, 37–46.

Brendan Jennings and Rolf Stadler. 2014. Resource management in clouds: Survey and research challenges. *Journal of Network and Systems Management*, 1–53.

Gueyoung Jung, Galen Swint, Jason Parekh, Calton Pu, and Akhil Sahai. 2006. Detecting bottleneck in n-tier it applications through analysis. In *Large Scale Management of Distributed Systems*. Springer, 149–160.

Srikanth Kandula, Ratul Mahajan, Patrick Verkaik, Sharad Agarwal, Jitendra Padhye, and Victor Bahl. 2009. Detailed diagnosis in computer networks. In *ACM SIGCOMM*.

Hui Kang, Xiaoyun Zhu, and Jennifer L. Wong. 2012. DAPA: diagnosing application performance anomalies for virtualized infrastructures. In *Presented as part of the 2nd USENIX Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services*. USENIX.

Terence Kelly. 2005a. Detecting performance anomalies in global applications. In *Proceedings of the 2nd Workshop on Real, Large Distributed Systems (WORLDS'05)*.

Terence Kelly. 2005b. Transaction mix performance models: Methods and application to performance anomaly detection. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles*. ACM, 1–3.

Kissmetrics. 2014. How Loading Time Affects Your Bottom Line. Retrieved April 15, 2014 from http://blog.kissmetrics.com/loading-time/.

David Kleinbaum, Lawrence Kupper, Azhar Nizam, and Eli Rosenberg. 2013. *Applied Regression Analysis and Other Multivariable Methods*. Cengage Learning.

Seth Koehler, Greg Stitt, and Alan D. George. 2011. Platform-aware bottleneck detection for reconfigurable computing applications. *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 4, 3, 30.

S. B. Kotsiantis. 2007. Supervised Machine Learning: A review of classification techniques. *Informatica* 31, 249–268.

Zhiling Lan, Ziming Zheng, and Yawei Li. 2010. Toward automated anomaly identification in large-scale systems. *IEEE Transactions on Parallel and Distributed Systems* 21, 2, 174–187.

Aleksandar Lazarevic, Levent Ertöz, Vipin Kumar, Aysel Ozgur, and Jaideep Srivastava. 2003. A comparative study of anomaly detection schemes in network intrusion detection. In *Proceedings of SIAM International Conference on Data Mining*. SIAM, 25–36.

Aleksandar Lazarevic, Nisheeth Srivastava, Ashutosh Tiwari, Josh Isom, Nikunj C. Oza, and Jaideep Srivastava. 2009. Theoretically optimal distributed anomaly detection. In *Proceedings of the IEEE International Conference on Data Mining Workshops (ICDMW'09)*. IEEE, 515–520.

Benjamin C. Lee and David M. Brooks. 2006. Accurate and efficient regression modeling for microarchitectural performance and power prediction. In *ACM SIGPLAN Notices*, Vol. 41. ACM, 185–194.

Donghun Lee, Sang K. Cha, and Arthur H. Lee. 2012. A performance anomaly detection and analysis framework for DBMS development. *IEEE Transactions on Knowledge and Data Engineering* 24, 8, 1345–1360.

Han Bok Lee and Benjamin G. Zorn. 1997. BIT: A Tool for instrumenting java bytecodes. In *Proceedings of the USENIX Symposium on Internet Technologies and Systems*. 73–82.

Wenke Lee and Dong Xiang. 2001. Information-theoretic measures for anomaly detection. In *Proceedings of IEEE Symposium on Security and Privacy (S&P'01)*. IEEE, 130–143.

Li Li and Allen D. Malony. 2006. Model-based performance diagnosis of master-worker parallel computations. In *Euro-Par 2006 Parallel Processing*. Springer, 35–46.

Yihua Liao and V. Rao Vemuri. 2002. Use of k-nearest neighbor classifier for intrusion detection. *Computers & Security* 21, 5, 439–448.

David J. Lilja. 2005. *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press.

Joao Paulo Magalhaes and L. Moura Silva. 2011. Adaptive profiling for root-cause analysis of performance anomalies in web-based applications. In *Proceedings of the 10th IEEE International Symposium on Network Computing and Applications (NCA'11)*. IEEE, 171–178.

Joao Paulo Magalhaes and Luis Moura Silva. 2010. Detection of performance anomalies in web-based applications. In *Proceedings of the 9th IEEE International Symposium on Network Computing and Applications (NCA'10)*. IEEE, 60–67.

João Paulo Magalhães and Luis Moura Silva. 2011. Root-cause analysis of performance anomalies in web-based applications. In *Proceedings of the 2011 ACM Symposium on Applied Computing*. ACM, 209–216.

Nihar R. Mahapatra and Balakrishna Venkatrao. 1999. The processor-memory bottleneck: Problems and solutions. *Crossroads* 5, 3es, 2.

Simon Malkowski, Markus Hedwig, Jason Parekh, Calton Pu, and Akhil Sahai. 2007. Bottleneck detection using statistical intervention analysis. In *Managing Virtualization of Networks and Services*. Springer, 122–134.

Simon Malkowski, Markus Hedwig, and Calton Pu. 2009. Experimental evaluation of N-tier systems: Observation and analysis of multi-bottlenecks. In *Proceedings of the IEEE International Symposium on Workload Characterization (IISWC'09)*. IEEE, 118–127.

Markos Markou and Sameer Singh. 2003. Novelty detection: A review part 1: Statistical approaches. *Signal processing* 83, 12, 2481–2497.

Andrew McHugh. 2013. Top 10 Web Outages of 2013. Retrieved March 11, 2015 from http://blog.smartbear.com/performance/top-10-web-outages-of-2013/.

Bob Melander, Mats Bjorkman, and Per Gunningberg. 2000. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *Proceedings of the Global Telecommunications Conference (GLOBECOM'00). IEEE*, Vol. 1. IEEE, 415–420.

Ningfang Mi, Giuliano Casale, Ludmila Cherkasova, and Evgenia Smirni. 2008a. Burstiness in multi-tier applications: Symptoms, causes, and new models. In *Proceedings of the 9th ACM/IFIP/USENIX International Conference on Middleware*. Springer-Verlag, New York, 265–286.

Ningfang Mi, Ludmila Cherkasova, Kivanc Ozonat, Julie Symons, and Evgenia Smirni. 2008b. Analysis of application performance and its change via representative application signatures. In *Proceedings of the Network Operations and Management Symposium*. IEEE, 216–223.

Jogesh K. Muppala, Steven P. Woolet, and Kishor S. Trivedi. 1991. Real-time systems performance in the presence of failures. *Computer* 24, 5, 37–47.

A. S. Navaz, V. Sangeetha, and C. Prabhadevi. 2013. Entropy based anomaly detection system to prevent ddos attacks in cloud. *International Journal of Computer Applications (0975-8887)* 62, 15. http://arxiv.org/abs/1308.6745

John E. Neilson, C. Murray Woodside, Dorina C. Petriu, and Shikharesh Majumdar. 1995. Software bottlenecking in client-server systems and rendezvous networks. *IEEE Transactions on Software Engineering* 21, 9, 776–782.

Hiep Nguyen, Zhiming Shen, Yongmin Tan, and Xiaohui Gu. 2013. FChain: Toward black-box online fault localization for cloud systems. In *Proceedings of the IEEE 33rd International Conference on Distributed Computing Systems (ICDCS'13)*. IEEE, 21–30.

George Nychis, Vyas Sekar, David G. Andersen, Hyong Kim, and Hui Zhang. 2008. An empirical evaluation of entropy-based traffic anomaly detection. In *Proceedings of the 8th ACM SIGCOMM conference on Internet Measurement*. ACM, 151–156.

John S. Oakland. 2008. *Statistical Process control*. Routledge.

Husanbir S. Pannu, Jianguo Liu, and Song Fu. 2012. A self-evolving anomaly detection framework for developing highly dependable utility clouds. In *Proceedings of the Global Communications Conference (GLOBECOM'12)*. IEEE, 1605–1610.

Iakovos Panourgias. 2011. *NUMA Effects on Multicore, Multisocket Systems*. The University of Edinburgh.

Jason Parekh, Gueyoung Jung, Galen Swint, Calton Pu, and Akhil Sahai. 2006. Issues in bottleneck detection in multi-tier enterprise applications. In *Proceedings of the 14th IEEE International Workshop on Quality of Service (IWQoS'06)*. IEEE, 302–303.

Emanuel Parzen. 1962. On estimation of a probability density function and mode. *The Annals of Mathematical Statistics*, 1065–1076.

Johannes Passing. 2005. Profiling, monitoring and tracing in SAP web application server. Seminar Systems Modelling, Hasso Plattner Insitute for Software Systems Engineering.

Manjula Peiris, James H. Hill, Jorgen Thelin, Sergey Bykov, Gabriel Kliot, and Christian Konig. 2014. PAD: Performance anomaly detection in multi-server distributed systems. In *Proceedings of the 7th IEEE International Conference on Cloud Computing (CLOUD'14)*. IEEE.

Soila Pertet and Priya Narasimhan. 2005. Causes of failure in web applications (cmu-pdl-05-109). *Parallel Data Laboratory*, 48.

Rob Powers, Moises Goldszmidt, and Ira Cohen. 2005. Short term performance forecasting in enterprise systems. In *Proceedings of the 11th ACM SIGKDD International Conference on Knowledge Discovery in Data Mining*. ACM, 801–807.

Calton Pu, Akhil Sahai, Jason Parekh, Gueyoung Jung, Ji Bae, You-Kyung Cha, Timothy Garcia, Danesh Irani, Jae Lee, and Qifeng Lin. 2007. An observation-based approach to performance characterization of distributed n-tier applications. In *IEEE 10th International Symposium on Workload Characterization. IISWC 2007*. IEEE, 161–170.

Xing Pu, Ling Liu, Yiduo Mei, Sankaran Sivathanu, Younggyun Koh, and Calton Pu. 2010. Understanding performance interference of i/o workload in virtualized cloud environments. In *Proceedings of the IEEE 3rd International Conference on Cloud Computing (CLOUD'10)*. IEEE, 51–58.

Sutharshan Rajasegarar, Christopher Leckie, and Marimuthu Palaniswami. 2008. Anomaly detection in wireless sensor networks. *Wireless Communications, IEEE* 15, 4, 34–40.

Christoph Rathfelder, Stefan Becker, Klaus Krogmann, and Ralf Reussner. 2012. Workload-aware system monitoring using performance predictions applied to a large-scale e-mail system. In *Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA'12)*. IEEE, 31–40.

Charles Reiss, John Wilkes, and Joseph L. Hellerstein. 2011. Google cluster-usage traces: Format+ schema. *Google Inc., White Paper*.

Douglas Reynolds. 2009. Gaussian mixture models. *Encyclopedia of Biometrics*, 659–663.

S. Rogers and M. Girolami. 2011. *A First Course in Machine Learning*. Taylor & Francis.

Raja R. Sambasivan, Alice X. Zheng, Michael De Rosa, Elie Krevat, Spencer Whitman, Michael Stroucken, William Wang, Lianghong Xu, and Gregory R. Ganger. 2011. Diagnosing performance changes by comparing request flows. In *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*. USENIX Association, 43–56.

Bianca Schroeder, Garth Gibson, and others. 2010. A Large-scale study of failures in high-performance-computing systems. *IEEE Transactions on Dependable and Secure Computing* 7, 4, 337–350.

Craig A. Shallahamer. 1995. Predicting Computing System Capacity and Throughput. Oracle Corporation White Paper. Retrieved from http://www.orapub.com.

Claude Elwood Shannon. 2001. A mathematical theory of communication. *ACM SIGMOBILE Mobile Computing and Communications Review* 5, 1, 3–55.

Bikash Sharma, Praveen Jayachandran, Akshat Verma, and Chita R. Das. 2013. CloudPD: Problem determination and diagnosis in shared dynamic clouds. In *Proceedings of the 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'13)*. IEEE, 1–12.

Sameer Shende. 1999. Profiling and tracing in Linux. In *Proceedings of the Extreme Linux Workshop*, Vol. 2. Citeseer.

Derek Smith, Qiang Guan, and Song Fu. 2010. An anomaly detection framework for autonomic management of compute cloud systems. In *Proceedings of the IEEE 34th Annual Computer Software and Applications Conference Workshops (COMPSACW'10)*. IEEE, 376–381.

Ralf Steuer, Jürgen Kurths, Carsten O. Daub, Janko Weise, and Joachim Selbig. 2002. The mutual information: Detecting and evaluating dependencies between variables. *Bioinformatics* 18, Suppl 2, S231–S240.

Yongmin Tan and Xiaohui Helen Adviser-Gu. 2012. *Online Performance Anomaly Prediction and Prevention for Complex Distributed Systems*. North Carolina State University.

Yongmin Tan and Xiaohui Gu. 2010. On predictability of system anomalies in real world. In *Proceedings of the IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS'10)*. IEEE, 133–140.

Yongmin Tan, Xiaohui Gu, and Haixun Wang. 2010. Adaptive system anomaly prediction for large-scale hosting infrastructures. In *Proceedings of the 29th ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing*. ACM, 173–182.

Yongmin Tan, Hiep Nguyen, Zhiming Shen, Xiaohui Gu, Chitra Venkatramani, and Deepak Rajan. 2012. Prepare: Predictive performance anomaly prevention for virtualized cloud systems. In *Proceedings of the IEEE 32nd International Conference on Distributed Computing Systems (ICDCS'12)*. IEEE, 285–294.

Jean-Claude Tarby, Houcine Ezzedine, José Rouillard, Chi Dung Tran, Philippe Laporte, and Christophe Kolski. 2007. Traces using aspect oriented programming and interactive agent-based architecture for early usability evaluation: Basic principles and comparison. In *Human-Computer Interaction. Interaction Design and Usability*. Springer, 632–641.

Igor Trubin. 2005. Capturing workload pathology by statistical exception detection system. In *Proceedings of the Computer Measurement Group*. Citeseer.

Igor A. Trubin and Linwood Merritt. 2004. Mainframe global and workload level statistical exception detection system, based on MASF. In *Proceedings of the International CMG Conference*. 671–678.

John Wilder. 1977. Exploratory data analysis. Addison-Wesley, Reading, Mass.

Arno Wagner and Bernhard Plattner. 2005. Entropy based worm and anomaly detection in fast IP networks. In *14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise*. IEEE, 172–177.

Christian Walck. 2007. Handbook on statistical distributions for experimentalists. Internal Report SUF-PFY/96-01, University of Stockholm.

Chengwei Wang, Karsten Schwan, and Matthew Wolf. 2009. Ebat: An entropy based online anomaly tester for data center management. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management-Workshops*. IEEE, 79–80.

Chengwei Wang, Vanish Talwar, Karsten Schwan, and Parthasarathy Ranganathan. 2010. Online detection of utility cloud anomalies using metric distributions. In *Proceedings of the Network Operations and Management Symposium (NOMS'10)*. IEEE, 96–103.

Chengwei Wang, Krishnamurthy Viswanathan, Lakshminarayan Choudur, Vanish Talwar, Wade Satterfield, and Karsten Schwan. 2011. Statistical techniques for online anomaly detection in data centers. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM'11)*. IEEE, 385–392.

Haichuan Wang, Qiming Teng, Xiao Zhong, and Peter F. Sweeney. 2009. Understanding cross-tier delay of multi-tier application using selective invocation context extraction. In *Proceedings of the 10th ACM/IFIP/USENIX International Conference on Middleware*. Springer-Verlag, New York, 34.

Qingyang Wang, Yasuhiko Kanemasa, Jack Li, Deepal Jayasinghe, Toshihiro Shimizu, Masazumi Matsubara, Motoyuki Kawaba, and Calton Pu. 2013a. Detecting transient bottlenecks in n-tier applications through fine-grained analysis. In *Proceedings of the IEEE 33rd International Conference on Distributed Computing Systems (ICDCS'13)*. IEEE, 31–40.

Qingyang Wang, Yasuhiko Kanemasa, Jack Li, Deepal Jayasinghe, Toshihiro Shimizu, Masazumi Matsubara, Motoyuki Kawaba, and Calton Pu. 2013b. An experimental study of rapidly alternating bottlenecks in n-tier applications. In *Proceedings of the IEEE 6th International Conference on Cloud Computing (CLOUD'13)*. IEEE, 171–178.

Tao Wang, Jun Wei, Feng Qin, WenBo Zhang, Hua Zhong, and Tao Huang. 2013. Detecting performance anomaly with correlation analysis for Internetware. *Science China Information Sciences* 56, 8, 1–15.

Tao Wang, Jun Wei, Wenbo Zhang, Hua Zhong, and Tao Huang. 2014. Workload-aware anomaly detection for web applications. *Journal of Systems and Software* 89, 19–32.

Tao Wang, Wenbo Zhang, Jun Wei, and Hua Zhong. 2012. Workload-aware online anomaly detection in enterprise applications with local outlier factor. In *Proceedings of the IEEE 36th Annual Computer Software and Applications Conference (COMPSAC'12)*. IEEE, 25–34.

Pengcheng Xiong, Calton Pu, Xiaoyun Zhu, and Rean Griffith. 2013. vPerfGuard: An automated model-driven framework for application performance diagnosis in consolidated cloud environments. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering*. ACM, 271–282.

Yahoo! 2014. Webscope dataset—Computer System Data. Retrieved from http://webscope.sandbox.yahoo.com/catalog.php?datatype=s.

Lingyun Yang, Chuang Liu, Jennifer M. Schopf, and Ian Foster. 2007. Anomaly detection and diagnosis in grid environments. In *Proceedings of the 2007 ACM/IEEE Conference on Supercomputing*. IEEE, 1–9.

Li Yu and Zhiling Lan. 2013. A scalable, non-parametric anomaly detection framework for Hadoop. In *Proceedings of the 2013 ACM Cloud and Autonomic Computing Conference*. ACM, 22.

Minlan Yu, Albert Greenberg, Dave Maltz, Jennifer Rexford, Lihua Yuan, Srikanth Kandula, and Changhoon Kim. 2011. Profiling network performance for multi-tier data center applications. In *Proceedings of Symposium on Networked System Design and Implementation*. 57–70.

Qi Zhang, Lu Cheng, and Raouf Boutaba. 2010. Cloud computing: State-of-the-art and research challenges. *Journal of Internet Services and Applications* 1, 1, 7–18.

Qi Zhang, Ludmila Cherkasova, Guy Mathews, Wayne Greene, and Evgenia Smirni. 2007b. R-capriccio: A capacity planning and anomaly detection tool for enterprise services with live workloads. In *Middleware 2007*. Springer, 244–265.

Qi Zhang, Ludmila Cherkasova, and Evgenia Smirni. 2007a. A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In *Proceedings of the 4th International Conference on Autonomic Computing (ICAC'07)*. IEEE, 27–27.

Steve Zhang, Ira Cohen, Moises Goldszmidt, Julie Symons, and Armando Fox. 2005. Ensembles of models for automated diagnosis of system performance problems. In *Proceedings of International Conference on Dependable Systems and Networks*. IEEE, 644–653.

Received December 2014; revised March 2015; accepted May 2015