Performance Anomaly Detection and Bottleneck Identification

Article *in* ACM Computing Surveys · June 2015

DOI: 10.1145/2791120

Olumuyiwa Ibidunmoye
Umeå University
12 PUBLICATIONS **29** CITATIONS
SEE PROFILE

Erik Elmroth
Umeå University
159 PUBLICATIONS **3,125** CITATIONS
SEE PROFILE

Performance Anomaly Detection and Bottleneck Identification [*]

Olumuyiwa Ibidunmoye
Francisco Hernández-Rodriguez
Erik Elmroth
{*muyi,francisco,elmroth*}*@cs.umu.se*
Umeå University, Sweden

July 3, 2015

Abstract

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. [1]

[*] ©ACM, 2015. This is the author's version of the work. It is posted here by permission of ACM for your personal use. Not for redistribution. The definitive version was published in ACM Computing Surveys, VOL. 48, ISS. 1, June 2015 http://dx.doi.org/10.1145/2791120

[1] **ACM Reference Format:** Olumuyiwa Ibidunmoye, Francisco Hernandez-Rodriguez, and Erik Elmroth. 2015. Performance anomaly detection and bottleneck identification. ACM Comput. Surv. 48, 1, Article 4 (June 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 [1]. 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, hardware failure, have been found to be sources of performance degradation in large scale systems [[2]; [3]]. Regardless of the sources of the problem, the challenge is how to detect performance anomalies, and to identify potential root-causes. The scale, dynamics, and heterogeneity of today's IT infrastructure further aggravate the problem.

Performance bottlenecks and anomalies are barriers to achieving predictable performance guarantees in enterprise applications and often come with significant cost implications. Studies [4] 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 [5], Amazon, experiences 1% decrease in sales for additional 100 ms delay in response time while Google reports a 20% drop in traffic due to 500 ms 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, performance bottlenecks if left unattended, 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 [1]. Global web services such as Yahoo Mail, Amazon Web Services, Google, LinkedIn, and Facebook have recently suffered from such failures [6]. Unplanned downtimes have significant cost implications [7] 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 [8]

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 [9]. 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 paper in Section 1. Section 2 presents a background of the problem, 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 [10].

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) [10].

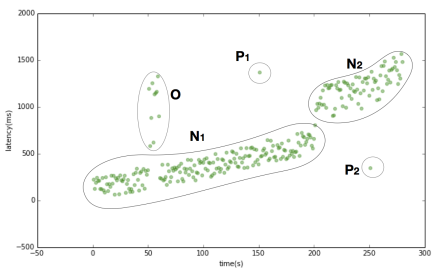
System *resources* includes physical components such as the CPU, memory, disk, caches, network and virtual components such as the network connections (e.g. sockets), locks, file handles or descriptors [10]. Resource *capacity* describes the storage size or processing strength of a given resource such 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 [[11]; [12]].

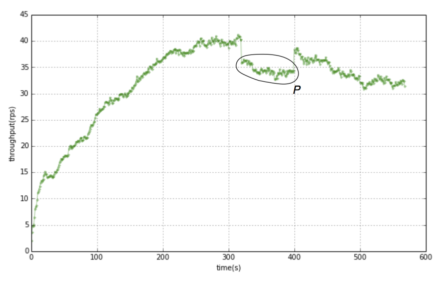
In the following sections we present various aspects of the PADBI problem.

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 [13]. In performance studies the data points are discrete measurements of a performance metric, throughput for example. Fig. 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. Fig. 1(b) captures another example of throughput anomaly, the group of points P represent a short dip in system throughput.



(a) Latency anomaly.



(b) Throughput anomaly.

Figure 1: Illustration of anomalies.

2.2.1 Types of Anomalies

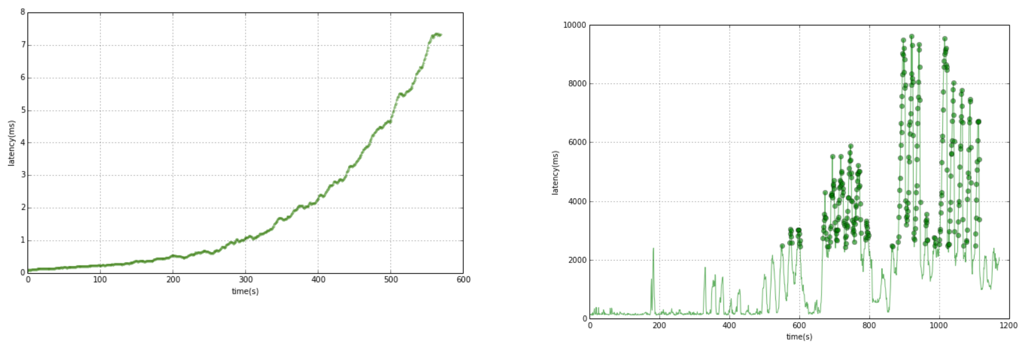
Chandola et al [9] identifies three basic types of anomalies; *point*, *collective* and *contextual* anomalies. These types only capture anomaly in terms of individual or contiguous data points, however, performance metrics are also known to commonly exhibit characteristic shapes when a resource is saturated [14]. Therefore we present one more type of anomalies, the *pattern* anomaly, which characterizes performance behaviours in terms of the structure or shapes of their curves rather than finite data points [15].

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 Fig. 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. Fig. 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 Fig. 1, the group of points labeled O in Fig. 1(a) and points labeled P in Fig. 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) etc.



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

Figure 2: Latency anomalies.

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 [15]. For example, application latency is known to exhibit an asymptotic growth as seen in Fig. 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 since the anomalous shape is made up of a set of data points that are as well collective anomalies.

2.3 Performance Bottlenecks

A bottleneck is a resource or an application component that limits the performance of a system [10]. Malkowski et al [16] describes 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 [17]. Such components often exhibit frequent congestion of load [18]. Also, application or system metrics correlating with an observed performance limitation are referred to as bottleneck metrics [19].

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 [10]. For example, Fig. 3 depicts a saturated CPU past the threshold usage level of 70%. According to Gregg [10] saturation may also 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

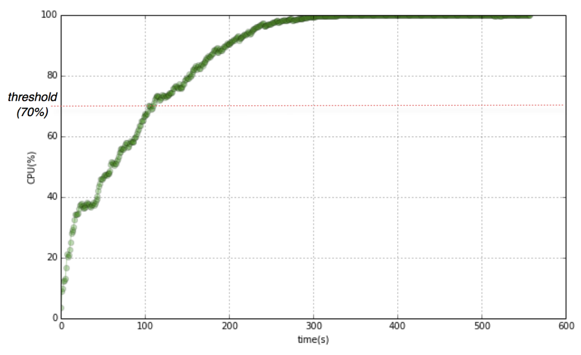


Figure 3: CPU saturation bottleneck.

caused by *memory leaks*[2] 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 multi-tasking environments, application processes contend for limited system resources such as CPU cycles, IO bandwidth, physical memory, and also software resources such as buffers, queues, semaphores and mutexes. The impact of such contention is well pronounced in cloud datacenters due to resource interference between multiple cloud tenants. The *noisy neighbours* effect is an analogy for this interference [20]. Several contention scenarios are well known for different system resources. CPU Contention— multiplexing the CPU between multiple processes causes frequent congested queue and performance interference in virtualized systems especially in the presence of CPU *hogging*[3] programs. Memory Contention — sharing limited memory bandwidth and processor-memory interconnect among processes may result in significant performance impact. Disk Contention— the processor-IO performance gap and restricted disk payload[4] causing substantial performance loss especially in IO workloads. Network Contention— excess demands for communication links at peak times lowers the effective bandwidth offered resulting in undesirable network contention delays.[5]

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. [16].

2. *Multiple Bottlenecks.* Two or more system resources or component may saturate simultaneously, or concurrently due to interdependency. Malkowski et al [16] 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

Fig. 4 is an extended *Fish-bone* diagram explaining the inter-relationships 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

[2]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.

[3]CPU hogging programs place excessive demand on compute resources thereby impacting the performance of other applications on the same host.

[4]Disk payload is in terms of size (in bytes) and number of IO requests (read/write) per second.

[5]Network contention delay is expressed as ratio of possible demand for a given network link to its maximum capacity.

the primary causes. Secondary causes that further explain primary causes and effects are represented with red slanted arrows. The thick horizontal arrow, the *spline*, depicts how primary and secondary causes can be used to explain main effects from left to right.

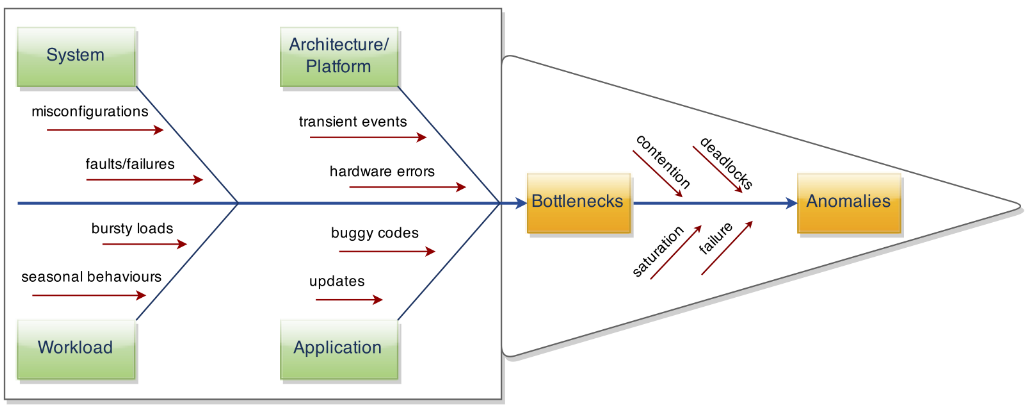


Figure 4: Cause-Effect Relationships of Performance Anomalies & Bottlenecks

2.4.1 Application Issues

Application level issues such as incorrect tunning, buggy codes, and software updates are examples of bottleneck sources [21]. Incorrect application configuration and updates may introduce unexpected resource bottlenecks [2].

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*[6] culminate into workload burstiness [18]. The undesirable effects of such load behaviour include congested queues, oversubscribed threading resources, present of short and uneven peaks in resource and performance measurements [22].

2.4.3 Architectures & 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 [23] and Mi et al [18] have shown how JVM garbage collection and Intel SpeedStep technology can induce bottlenecks. In modern systems with multi-core NUMA architecture, the location of memory relative to a processor may affect application performance especially those with memory-bound workloads [24].

2.4.4 System Faults

Faults in system resources and components may considerably affect application performance [25] 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 [1].

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 non-trivial. 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 [26].

[6]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 etc.

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. It is therefore hard to precisely distinguish normal application behaviours from anomalous behaviours at runtime [27].

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 the data in a uniform manner [27]. 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 1 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 9 and 10 of Appendix A.

Table 1: Recent literature by goals

PAD	**PBI**	**PADBI**
[28]		
[29]		
[30]		[48]
[31]		[21]
[27]	[42]	[49]
[32]	[43]	[50]
[33]	[44]	[51]
[34]	[45]	[16]
[35]	[46]	[52]
[36]	[47]	[53]
[37]	[26]	[17]
[38]		[54]
[39]		[55]
[40]		
[41]		

Generally, the output of PADBI systems may include a set of anomalous performance *indices*, a *time-stamp* (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 *Positive Predictive Value* (PPV) in literature.

Table 2: Recent literature on systems

Reference	Dedicated Server	Virtualized, Cloud	Grid	Distributed	Web	Multi-tier
[41]				✓	✓	✓
[55]		✓				
[56]				✓		
[47]		✓		✓	✓	✓
[58]		✓		✓		
[59]		✓			✓	
[38]				✓		
[60]		✓		✓	✓	✓
[39]		✓		✓	✓	✓
[61]		✓		✓	✓	✓
[62]		✓		✓		
[63]		✓				✓
[64]		✓		✓	✓	
[65]	✓			✓	✓	
[54]		✓		✓		✓
[53]	✓			✓	✓	✓
[66]		✓		✓	✓	✓
[67]		✓		✓	✓	✓
[27]	✓					
[45]		✓			✓	
[51]		✓		✓		
[68]	✓			✓		
[18]		✓		✓		✓
[29]			✓	✓		
[30]			✓	✓		
[43]	✓			✓	✓	✓
[50]	✓			✓	✓	✓

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 *Sensitivity* 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 multi-tier 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 [56]. While the Grid [30] is known for running short- and long-term applications performing large computations across distributed nodes, cloud infrastructures [57] 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 inter-dependencies 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 datacenters. This complex composition and deployment brings special requirements for localization of performance problems.

Table 2 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 closely related set of 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*— these 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*— these 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*, *CPU queue length* etc.

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 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 be profiled in a non-intrusive manner [60].

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 [69]. 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 mis-behaviour 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 [70]. Runtime code instrumentation is a common tracing method. Tracing platforms such as Aspect Oriented Programming [71] and Java Byte Code instrumentation [[72];[73]] 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 [53]. 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 [39]. 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, 3.1.2, 3.1.3 and 3.1.4. References of research contributions in each category can be found in tables 9 and 10 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 run-time characteristics are called *signatures*, *fingerprints* or *profiles*. PADBI systems generate signatures as compact run-time 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 consume 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 [[74]; [75]; [76]; [77]; [2]].

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. Secondly, 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 real time systems because the experiential knowledge of right thresholds, transient anomaly behaviours have to be encoded into an automatic mechanism [[78]; [53]; [79]].

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 [[44]; [80]; [81]].

3.1.4 Flow and Dependency Analysis

By studying the flow of communication across components in distributed applications, performance anomalies and hot-spots 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. [[45]; [33]; [50]; [82]; [60]]

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 [[30]; [16]]. 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. Non-parametric 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 [[9]; [83]]. The *Median*, *CUSUM*, *Spearman correlation*, *Kruskal-Wallis* and *Wilcoxon's* tests are examples of non-parametric statistics [84].

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. $X \sim N(\mu, \sigma^2)$) [[9]; [85]].

The Tukey [86] 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 inter quantile 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 [87].

The density distribution may also be exploited for detecting anomalous data points based on the Gaussian Mixture Model (GMM) [85]. GMMs are parametric models of the probability distribution of continuous random variables estimated using the iterative Expectation-

Table 3: Literature on regression-based approaches.

Author	**Methodology**
[21]	Presents a Regression-based TM model for identifying performance anomalies in geographically distributed applications. Tested with commercial applications such as ACME, FT, and VDR.
[96]	Proposes an approach using Non-negative Least Square(NLS) regressive TM models for estimating resource demands by different client transactions and applicability in resource provisioning.
[31]	Presents Signature-based anomaly detection built on [96]
[2]	Demonstrates how stepwise linear regression addresses model "overfitting" problem in [96] and further present a segmentation-based method (as an extension of [31]) for detecting performance changes in enterprise web applications.
[54]	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.
[30]	Models the relationship between application metrics and system metrics for metric selection, reduction and anomaly detection.

Maximization (EM) algorithm [88].
Given a data set X composed of n normally distributed features $\{x^{(1)}, x^{(2)}, ..., x^{(n)}\}$ with corresponding parameters $\{\mu_1, \mu_2, ..., \mu_n\}$ and $\{\sigma_1^2, \sigma_2^2, ..., \sigma_n^2\}$, 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 $X = \{x^{(1)}, x^{(2)}, ..., x^{(n)}\}$ 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 i.e. $P(X) = \prod_{i=1}^{n} P(x^{(i)}; \mu_i, {\sigma_i}^2)$. The value of ϵ can be varied depending on application requirements [[89]; [90]].
Other density-based methods include the *Parzen* Windows Estimation [91], the *Grubb's* test [92] and the *Student's t*-test [85].

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 non-linear) 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)* [93].

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 [9]. New observations falling outside the confidence interval produced by the model may be classified as anomalous [[94]; [95]]. 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 3 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

Table 4: Literature on Correlation-based approaches.

Author	**Methodology**
[50]	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.
[3] [53]	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.
[54]	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.
[39]	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.
[59] [67]	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.
[98]	Presents a novel Kriging-based model of system performance as a function of dynamic resource allocation and workloads to predict and detect performance problems.
[56]	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.

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 [3]. 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 window $[t_1, t_n]$ in the form $W_a = \{a_1, a_2, .., a_n\}$ and $W_b = \{b_1, b_2, .., b_n\}$ 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 $X = \{x_1, x_2, x_3, ...x_n\}$ and $Y = \{y_1, y_2, y_3, ...y_n\}$, 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 [97] is a popular variant of this method. See Table 4 for references relating to correlation-based systems.

3.2.1.4 Statistical Process Control Statistical process control (SPC) [99], 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 [100], 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, [101] 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

Table 5: Literature on other statistical methods.

Method	Highlight	Reference
ANOVA	Root-cause identification, and Change detection	[52] [100]
Index of Dispersion	Workload burstiness and variability detection in web applications.	[18] [63]
Mean Standard Deviation, Cummulative Density Function	Anomaly detection in Grid application.	[29]
Student's t-test	Localization of anomalous metrics.	[41]
Markov Model	Prediction of anomalous performance metrics and fault localization.	[60] [34] [103] [68]
Kernel Density Estimation	Estimation density functions using Kernel regression for inferring resource saturation.	[16]
Probability Models	Anomaly prediction and detection.	[48], [104], [51], [79]
Statistical Process Control	Detection fo performance transient and persistent anomalies and identification of anomalous correlation in database and enterprise systems.	[105] [106] [107] [17] [59] [100]
Statistical Intervention Analysis	Bottleneck identification.	[43]

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 [87].

3.2.1.5 Statistical Intervention Analysis Statistical Intervention Analysis (SIA) [102] measures the form and magnitude of shifts in time series 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 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 [108]. 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 [109].

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 datacenters 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 nor 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 since no training is involved. And are particularly suitable for detecting unknown anomalies in cloud datacenters where precise definition of anomaly characteristics may not always exist.

Semi-supervised learning An emerging approach is to maximize the best of supervised and unsupervised learning. Semi-supervised algorithms assume a small chunk of the dataset is labeled usually the normal class and the remaining unlabeled instances are anomalous. They often out-perform 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 pre-processing tasks described below.

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 method for doing this. PCA takes k correlated metrics as input and reduces them to $m \leq k$ non-dependent metrics. These m metrics can be interpreted as linear combinations of the original set [[110]; [32]]. Other methods for dimension reduction include Factor analysis, Independent component analysis, and Non-linear PCA [111].

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 [[112]; [113]].

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 6, 7, and 8.
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 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* [118].

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 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

Table 6: Supervised PADBI Systems.

Reference	**Technique**	**Methodology**
[61] [79]	Bayesian classifier and Tree augmented networks (TAN).	Presents a method for predicting and classifying anomalies.
[32]	Decision trees and TAN.	Presents a technique for reducing metric dimensions based on Mutual Information and PCA and identifying performance anomalies Tree-based classifier.
[49]	Decision Tree	Presents a decision-tree based automated approach for detecting performance bottlenecks.
[48]	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.
[68]	Bayesian classification	A stream-based anomaly detection method is used to detect anomaly symptoms and infer their root-causes.
[19]	Tree augmented network, Bayesian networks, LogicBoost, C4.5 decision tree.	Explores the performance of various machine learning classifiers with regards to bottleneck detection in an enterprise applications.
[114]	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.

Table 7: Unsupervised PADBI Systems

Reference	**Technique**	**Methodology**
[41] [36]	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.
[62]	Self-Organizing Maps (SOM)	Presents an anomaly detection mechanism in IaaS Cloud using SOMs to learn emergent system behaviour and predict unknown anomalies.
[110]	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.
[40]	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.
[38]	Non-parametric Clustering	Proposes a decentralized approach for detecting anomalies in Hadoop clusters based on Hierarchical Grouping and majority voting.

Table 8: Semi-supervised PADBI Systems

Reference	**Technique**	**Methodology**
[27]	Principal and Independent Component Analysis	Presents an automated anomaly detection mechanisms for identifying system nodes whose behaviours are deviating from others in a cloud datacenter.
[35]	Classification, Clustering, Support Vector Machines	Presents a self-evolving mechanism for predicting and detecting of system failures in Cloud systems.
[115]	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.
[116]	K-Nearest Neighbours	Proposes an automated failure detection system employing distance-based anomaly rules to identify faulty machines in a cluster.
[37] [117]	Wavelets	Presents a method analyzing performance metrics in both time and frequency domains in order to identify anomalous behaviors in a Cloud environment.
[64]	Support Vector Machines (SVM)	Proposes an hybrid self-evolving anomaly detection framework using one-class and two-class SVM.

score is computed as the inverse of the confidence score associated with a given rule [9].

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 [9]. 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 kth-Nearest Neighbour which estimates the distance of a given instance to its nearest neighbours and evaluate the distance against a predefined domain specific threshold [[119]; [120]]. 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 [36]. 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 which groups similar data instances into clusters according to hidden relationships between instances in a cluster [121]. 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 [9]. The Euclidean distance, Mahalanobis, 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) [89]. 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 [122]. The primary assumption of these methods is that anomalies induce irregularities in the information content of a given data set [9]. Also they are very generic in nature with no need for parameterization [123]. The Entropy information measure or Shannon-Wiener Index [124] estimates the degree of uncertainty in a given data set. 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 [125]. 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 [126]. Entropy-based methods have been applied to study malicious behaviours in network traffic in [123] and [126]. Wang et al [127, 122] presents 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 [128] and suitable for online unsupervised detection of unknown anomalies [122] since no assumptions of underlying distribution is made.

4 Research Trends

Before the 2000's, contributions focused primarily on the detection of coarse-grained performance issues such as identifying hardware, software bottlenecks in the operating systems [[129]; [130]], networks [131] and client-server applications [132].

Due to the emergence of the Internet, the early 2000's witnessed a slow trend towards web and distributed applications hosted in dedicated environments. Chen et al [42], Aguilera et al [82], Barham et al [133] proposed techniques for uncovering performance failures and anomalies with regards to web and distributed systems. By mid to late 2000's efforts concentrated on building improved detection mechanisms targeting enterprise applications running in shared-hosting environment, grid and large scale infrastructures. This period witnessed the development of analytical approaches and tools such as; transaction mix models [134]; queuing-theoretic models [21]; signature models [[76]; [31]] and statistical techniques [[43]; [2]]. While efforts such as in [[49]; [16]] propose an experimental approach, [[44]; [50]] demonstrate the potential of analyzing the flow of messages across distributed components as a suitable method for detecting performance abnormalities.

From the late 2000's until now, the research contributions have been largely consolidated on achieving dependability [[37]; [17]], predictable performance [79], root-cause identification[[53]; [116]; [38]] and meeting performance guarantees [[54]; [27]] in cloud computing applications and systems. Similarly there are systems tailored to detecting and resolving workload related anomalies [[36]; [41]]. 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 [[62]; [55]; [40]; [39]]. Even though existing research contribution is dominated by reactive solutions, there is increasing shift towards 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 [[8]; [61]].

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 [57]. 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 [[135]; [136]]. The resulting resource contention and performance interference caused by resource sharing have significant impact on the performance of cloud services and systems [[39]; [122]].

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 below.

1. *Scale.* Medium to large scale cloud infrastructures run up to thousands of applications on limited computing resources. It is a daunting to keep track of the execution status of such huge applications base [[34]; [62]]. 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 [122] 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. *Multi-tenancy.* Multi-tenancy enables different applications (deployed in virtual machines (VMs)) to be co-located 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 an tight execution environment has been shown to account for 40% in performance degradation in some applications [39]. 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 [122]. 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 [[62]; [34]].

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 [[34]; [122]]. This leads to higher frequency of anomalies. Faulty VM reconfigurations, and spontaneous live migrations have been observed to impact performance by up to 30% and 10% respectively [39]. In such environments, it is nearly difficult to determine what performance behaviour is normal and which is not [62].

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 [[137]; [138]]. Delayed detection and manual resolutions does not fit the cloud model as they can cause prolonged performance violations with huge financial penalty and failure [[62]; [34]]. Therefore PADBI systems for the cloud are must be dynamic and proactive in nature [[39]; [122]].

6 Discussions & 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" [7] problem. Though parametric techniques assume known data distribution and best at identifying well known anomalies, non-parametric 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

[7] 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.

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, but 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 [64]. 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 trade-off 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.

Tables 9 and 10 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 below;

1. *Multi-level bottleneck detection.* Current efforts must extend towards the detection of performance bottlenecks at different levels considering the complexity of todays 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 etc) 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 towards this especially for common performance anomalies and bottlenecks. A similar direction is documented in [1].

3. *Open performance datasets.* There lack of open performance datasets hinders the pace of research in this area because such data are often considered highly sensitive or classified. The Google Cluster[139] 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 [140] project provides system measurements of the infrastructure running its cloud serving benchmark system [141]. 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 [142].

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 semi-automated 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 non-stationary cloud behaviours will greatly improve application performance. Tan et al [79], Tan and Gu [103], and Sharma et al [39] presents interesting directions in this case.

6. *Distributed detection.* Huge chunk of current research focus is on centralized detection. Modern enterprise systems are inherently distributed with components spanning

multiple physical domains (servers or datacenters). Often times the collection of data across such domains is impractical or difficult due to potential system overheads, proprietary and privacy regulations. This implication calls for a decentralized approach that fits naturally with such systems. A theoretical attempt is presented in [143] while a similar case study for failure detection is studied in [116].

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 multi-level bottleneck detection, distributed detection and accessible performance datasets still remain open research issues.

8 Acknowledgment

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).

References

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

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

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

[4] Kissmetrics. How loading time affects your bottom line, 2014.

[5] Cheng Huang. Public dns system and global traffic management, 2011.

[6] Andrew McHugh. Top 10 web outages of 2013, 2013.

[7] Evolven. Downtime, outages and failures - understanding their true costs, 2011.

[8] Qiang Guan, Ziming Zhang, and Song Fu. Proactive failure management by integrated unsupervised and semi-supervised learning for dependable cloud systems. In *Sixth International Conference on Availability, Reliability and Security (ARES)*, pages 83–90. IEEE, 2011.

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

[10] Brendan Gregg. *Systems Performance: Enterprise and the Cloud*. Pearson Education, 2013.

[11] Craig A Shallahamer. Predicting computing system capacity and throughput, 1995.

[12] David J Lilja. *Measuring computer performance: a practitioner's guide*. Cambridge University Press, 2005.

[13] Kaustav Das and Jeff Adviser-Schneider. Detecting patterns of anomalies. 2009.

[14] Neil J Gunther. Benchmarking blunders and things that go bump in the night. *CoRR*, 2004.

[15] Neil J Gunther. *Analyzing Computer System Performance with Perl:: PDQ*. Springer, 2011.

[16] Simon Malkowski, Markus Hedwig, and Calton Pu. Experimental evaluation of n-tier systems: Observation and analysis of multi-bottlenecks. In *IEEE International Symposium on Workload Characterization. IISWC 2009*, pages 118–127. IEEE, 2009.

[17] Donghun Lee, Sang K Cha, and Arthur H Lee. A performance anomaly detection and analysis framework for dbms development. *IEEE Transactions on Knowledge and Data Engineering*, 24(8):1345–1360, 2012.

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

[19] Jason Parekh, Gueyoung Jung, Galen Swint, Calton Pu, and Akhil Sahai. Issues in bottleneck detection in multi-tier enterprise applications. In *14th IEEE International Workshop on Quality of Service. IWQoS 2006*, pages 302–303. IEEE, 2006.

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

[21] Terence Kelly. Detecting performance anomalies in global applications. In *Second Workshop on Real, Large Distributed Systems (WORLDS2005)*, 2005.

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

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

[24] Iakovos Panourgias. Numa effects on multicore, multi socket systems. *The University of Edinburgh*, 2011.

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

[26] Qingyang Wang, Yasuhiko Kanemasa, Jack Li, Deepal Jayasinghe, Toshihiro Shimizu, Masazumi Matsubara, Motoyuki Kawaba, and Calton Pu. An experimental study of rapidly alternating bottlenecks in n-tier applications. In *IEEE Sixth International Conference on Cloud Computing (CLOUD)*, pages 171–178. IEEE, 2013.

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

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

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

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

[31] Ludmila Cherkasova, Kivanc Ozonat, Ningfang Mi, Julie Symons, and Evgenia Smirni. Anomaly? application change? or workload change? towards automated detection of application performance anomaly and change. In *IEEE International Conference on Dependable Systems and Networks With FTCS and DCC*, pages 452–461. IEEE, 2008.

[32] Song Fu. Performance metric selection for autonomic anomaly detection on cloud computing systems. In *Global Telecommunications Conference (GLOBECOM 2011)*, pages 1–5. IEEE, 2011.

[33] Raja R Sambasivan, Alice X Zheng, Michael De Rosa, Elie Krevat, Spencer Whitman, Michael Stroucken, William Wang, Lianghong Xu, and Gregory R Ganger. Diagnosing performance changes by comparing request flows. In *Proceedings of the 8th USENIX conference on Networked systems design and implementation*, pages 43–56. USENIX Association, 2011.

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

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

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

[37] Qiang Guan and Song Fu. Wavelet-based multi-scale anomaly identification in cloud computing systems. In *Global Communications Conference (GLOBECOM)*, pages 1379–1384. IEEE, 2013.

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

[39] Bikash Sharma, Praveen Jayachandran, Akshat Verma, and Chita R Das. Cloudpd: Problem determination and diagnosis in shared dynamic clouds. In *43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 1–12. IEEE, 2013.

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

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

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

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

[44] I-Hsin Chung, Guojing Cong, David Klepacki, Simone Sbaraglia, Seetharami Seelam, and Hui-Fang Wen. A framework for automated performance bottleneck detection. In *IEEE International Symposium on Parallel and Distributed Processing. IPDPS 2008*, pages 1–7. IEEE, 2008.

[45] Muli Ben-Yehuda, David Breitgand, Michael Factor, Hillel Kolodner, Valentin Kravtsov, and Dan Pelleg. Nap: a building block for remediating performance bottlenecks via black box network analysis. In *Proceedings of the 6th international conference on Autonomic computing*, pages 179–188. ACM, 2009.

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

[47] Pengcheng Xiong, Calton Pu, Xiaoyun Zhu, and Rean Griffith. vperfguard: an automated model-driven framework for application performance diagnosis in consolidated cloud environments. In *Proceedings of the ACM/SPEC international conference on International conference on performance engineering*, pages 271–282. ACM, 2013.

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

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

[50] Sandip Agarwala, Fernando Alegre, Karsten Schwan, and Jegannathan Mehalingham. E2eprof: Automated end-to-end performance management for enterprise systems, 2007.

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

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

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

[54] Hui Kang, Xiaoyun Zhu, and Jennifer L Wong. 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, 2012.

[55] Daniel J Dean, Hiep Nguyen, Peipei Wang, and Xiaohui Gu. 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*, pages 16–16. USENIX Association, 2014.

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

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

[58] Qiang Guan and Song Fu. Adaptive anomaly identification by exploring metric subspace in cloud computing infrastructures. In *IEEE 32nd International Symposium on Reliable Distributed Systems (SRDS)*, pages 205–214. IEEE, 2013.

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

[60] Hiep Nguyen, Zhiming Shen, Yongmin Tan, and Xiaohui Gu. Fchain: Toward black-box online fault localization for cloud systems. In *IEEE 33rd International Conference on Distributed Computing Systems (ICDCS)*, pages 21–30. IEEE, 2013.

[61] Yongmin Tan and Xiaohui Helen Adviser-Gu. *Online performance anomaly prediction and prevention for complex distributed systems*. North Carolina State University, 2012.

[62] Daniel Joseph Dean, Hiep Nguyen, and Xiaohui Gu. Ubl: unsupervised behavior learning for predicting performance anomalies in virtualized cloud systems. In *Proceedings of the 9th international conference on Autonomic computing*, pages 191–200. ACM, 2012.

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

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

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

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

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

[68] Xiaohui Gu and Haixun Wang. Online anomaly prediction for robust cluster systems. In *IEEE 25th International Conference on Data Engineering, 2009. ICDE'09*, pages 1000–1011. IEEE, 2009.

[69] Sameer Shende. Profiling and tracing in linux. In *Proceedings of the Extreme Linux Workshop*, volume 2. Citeseer, 1999.

[70] Johannes Passing. Profiling, monitoring and tracing in sap web application server. 2005.

[71] Jean-Claude Tarby, Houcine Ezzedine, José Rouillard, Chi Dung Tran, Philippe Laporte, and Christophe Kolski. 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*, pages 632–641. Springer, 2007.

[72] Han Bok Lee and Benjamin G Zorn. Bit: A tool for instrumenting java bytecodes. In *USENIX Symposium on Internet Technologies and Systems*, pages 73–82, 1997.

[73] Walter Binder, Jarle Hulaas, and Philippe Moret. Advanced java bytecode instrumentation. In *Proceedings of the 5th international symposium on Principles and practice of programming in Java*, pages 135–144. ACM, 2007.

[74] Peter Bodik, Moises Goldszmidt, Armando Fox, Dawn B Woodard, and Hans Andersen. Fingerprinting the datacenter: automated classification of performance crises. In *Proceedings of the 5th European conference on Computer systems*, pages 111–124. ACM, 2010.

[75] Peter Bodík, Moises Goldszmidt, and Armando Fox. Hilighter: Automatically building robust signatures of performance behavior for small-and large-scale systems. In *A. Fox and S. Basu, editors, SysML, USENIX Association*, 2008.

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

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

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

[79] Yongmin Tan, Xiaohui Gu, and Haixun Wang. Adaptive system anomaly prediction for large-scale hosting infrastructures. In *Proceedings of the 29th ACM SIGACT-SIGOPS symposium on Principles of distributed computing*, pages 173–182. ACM, 2010.

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

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

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

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

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

[85] Markos Markou and Sameer Singh. Novelty detection: a reviewpart 1: statistical approaches. *Signal processing*, 83(12):2481–2497, 2003.

[86] John W Tukey. Exploratory data analysis. 1977.

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

[88] Douglas Reynolds. Gaussian mixture models. *Encyclopedia of Biometrics*, pages 659–663, 2009.

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

[90] Christian Walck. Handbook on statistical distributions for experimentalists, 2007.

[91] Emanuel Parzen. On estimation of a probability density function and mode. *The annals of mathematical statistics*, pages 1065–1076, 1962.

[92] Frank E Grubbs. Procedures for detecting outlying observations in samples. *Technometrics*, 11(1):1–21, 1969.

[93] David Kleinbaum, Lawrence Kupper, Azhar Nizam, and Eli Rosenberg. *Applied regression analysis and other multivariable methods*. Cengage Learning, 2013.

[94] Marc Courtois and Murray Woodside. Using regression splines for software performance analysis. In *Proceedings of the 2nd international workshop on Software and performance*, pages 105–114. ACM, 2000.

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

[96] Qi Zhang, Ludmila Cherkasova, and Evgenia Smirni. A regression-based analytic model for dynamic resource provisioning of multi-tier applications. In *Fourth International Conference on Autonomic Computing. ICAC'07*, pages 27–27. IEEE, 2007.

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

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

[99] John S Oakland. *Statistical process control*. Routledge, 2008.

[100] Frank M Bereznay and Kaiser Permanente. Did something change? using statistical techniques to interpret service and resource metrics. In *Int. CMG Conference*, pages 229–242, 2006.

[101] Jeffrey P Buzen and Annie W Shum. Masf-multivariate adaptive statistical filtering. In *Int. CMG Conference*, pages 1–10, 1995.

[102] George EP Box and George C Tiao. Intervention analysis with applications to economic and environmental problems. *Journal of the American Statistical Association*, 70(349):70–79, 1975.

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

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

[105] Igor A Trubin and Linwood Merritt. " mainframe global and workload level statistical exception detection system, based on masf". In *Int. CMG Conference*, pages 671–678, 2004.

[106] Igor Trubin et al. Capturing workload pathology by statistical exception detection system. In *Proceedings of the Computer Measurement Group*. Citeseer, 2005.

[107] Jack Brey and Rick Sironi. Managing at the knee of the curve (the use of spc in managing a data center). In *Int. CMG Conference*, pages 895–901, 1990.

[108] S. Rogers and M. Girolami. *A First Course in Machine Learning*. Chapman & Hall/CRC machine learning & pattern recognition series. Taylor & Francis, 2011.

[109] E. Alpaydin. *Introduction to Machine Learning*. Adaptive Computation and Machine Learning Series. MIT Press, 2014.

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

[111] Imola K Fodor. A survey of dimension reduction techniques, 2002.

[112] Ralf Steuer, Jürgen Kurths, Carsten O Daub, Janko Weise, and Joachim Selbig. The mutual information: detecting and evaluating dependencies between variables. *Bioinformatics*, 18(suppl 2):S231–S240, 2002.

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

[114] Rob Powers, Moises Goldszmidt, and Ira Cohen. Short term performance forecasting in enterprise systems. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, pages 801–807. ACM, 2005.

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

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

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

[118] SB Kotsiantis. Supervised machine learning: A review of classification techniques. *Informatica*, 31:249–268, 2007.

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

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

[121] Pavel Berkhin. A survey of clustering data mining techniques. In *Grouping multidimensional data*, pages 25–71. Springer, 2006.

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

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

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

[125] AS Navaz, V Sangeetha, and C Prabhadevi. Entropy based anomaly detection system to prevent ddos attacks in cloud. *Int. Journal of Computer Applications (0975-8887)*, 62(15), 2013.

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

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

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

[129] Nihar R Mahapatra and Balakrishna Venkatrao. The processor-memory bottleneck: problems and solutions. *Crossroads*, 5(3es):2, 1999.

[130] John S Breese and Russ Blake. Automating computer bottleneck detection with belief nets. In *Proceedings of the Eleventh conference on Uncertainty in artificial intelligence*, pages 36–45. Morgan Kaufmann Publishers Inc., 1995.

[131] Bob Melander, Mats Bjorkman, and Per Gunningberg. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *Global Telecommunications Conference, 2000. GLOBECOM'00. IEEE*, volume 1, pages 415–420. IEEE, 2000.

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

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

[134] Terence Kelly. Transaction mix performance models: methods and application to performance anomaly detection. In *Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 1–3. ACM, 2005.

[135] Qi Zhang, Lu Cheng, and Raouf Boutaba. Cloud computing: state-of-the-art and research challenges. *Journal of internet services and applications*, 1(1):7–18, 2010.

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

[137] Rajkumar Buyya, Rodrigo N Calheiros, and Xiaorong Li. Autonomic cloud computing: Open challenges and architectural elements. In *Third International Conference on Emerging Applications of Information Technology (EAIT)*, pages 3–10. IEEE, 2012.

[138] Masum Z Hasan, Edgar Magana, Alexander Clemm, Lew Tucker, and Sree Lakshmi D Gudreddi. Integrated and autonomic cloud resource scaling. In *Network Operations and Management Symposium (NOMS)*, pages 1327–1334. IEEE, 2012.

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

[140] Yahoo! Webscope dataset— computer system data, 2014.

[141] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154. ACM, 2010.

[142] Bianca Schroeder and Garth Gibson. A large-scale study of failures in high-performance-computing systems (cmu-pdl-05-112). 2005.

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

[144] Haichuan Wang, Qiming Teng, Xiao Zhong, and Peter F Sweeney. 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*, page 34. Springer-Verlag New York, Inc., 2009.

A APPENDIX A

B General Overview of PADBI Systems

Table 9: Overview of PADBI Systems

Reference	Goal	System	Observability	Strategy	Method	Techniques
[42]	PBI	Distributed, Web-based, Component bottlenecks	White-box, Source Tracing	Flow & Dependency	Hybrid	Clustering, Correlation
[48]	PADBI	Multi-tier, Web-based, System metrics	Black-box, Profiling	Observational	Machine Learning	Tree Augmented Bayesian Networks
[21]	PAD	Distributed, Web-based, Application & System metrics	Gray-box, Profiling	Observational	Statistical	Regression, Transaction mix Model
[77]	PAD	Distributed, Enterprise, Application & System metrics	Black-box, Profiling	Signature-based	Machine Learning	Clustering, Tree-Augmented Naive Bayes Models
[49]	PADBI	Multi-tier, Application & System metrics	Black-box, Profiling	Observational	Machine Learning	C4.5 Decision Tree
[43]	PBI	Web-based, System metrics	Gray-box, Profiling	Knowledge-based	Statistical	SIA
[50]	PADBI	Distributed, Multi-tier, Application & System metrics	White-box	Flow & Dependency	Statistical	Correlation
[28]	PAD	Multi-tier, System & Application metrics	Gray, Profiling	Observational	Statistical	Regression, TM models, Queuing model
[29]	PAD	Grid, System metrics	Black-box, Logging	Observational	Statistical	MSD,CDF,EWMA
[30]	PAD	Grid, System resource metrics	Black-box, Profiling	Observation, Flow & Dependency	Statistical, Signal Processing	Extended Window Averaging, Regression
[44]	PBI	HPC, Resource bottlenecks	Gray, Profiling	Knowledge-based	-	Inference Engine
[31]	PAD	Web-based, Multi-tier, Application & System metrics	Gray, Profiling	Signature-based	Statistical	Regression, Transaction mix Model
[75]	PAD	Distributed Systems, System metrics	Black-box, Profiling	Signature-based	Machine Learning	Logistic Regression with L1 Regularization
[16]	PADBI	Multi-tier, Application & System metrics	Gray	Observational	Statistical, Signal Processing	Kernel Density Estimation, Adaptive Filtering
[144]	PBI	Multi-tier	Gray	Observational	-	Heuristics
[51]	PADBI	Enterprise Systems	Gray-box	Knowledge-based, Flow & Dependency	Statistical Learning	Probability Models, Inference Engine
[45]	PBI	Virtualized, Component & Resource bottlenecks	Black-box, Profiling	Flow & Dependency	Statistical, Queueing Theory	Percentile Testing, Little's Law
[46]	PBI	Cloud, Multi-tier, Resource bottlenecks	Black-box, Profiling	Observational	-	-
[74]	PAD	Distributed Systems, Cloud, System metrics	Black-box, Profiling	Signature-based	Statistical and Machine Learning	Quantile Summarization, Logistic Regression with L1 Regularization

Table 10: Overview of PADBI Systems (contd)

Reference	Goal	System	Observability	Strategy	Method	Techniques
[27]	PAD	Cloud, Host/Node bottlenecks	Black-box, Profiling	Observational	Machine Learning	Principal and Independent Component Analysis
[53]	PADBI	Web-based, Application metrics	White-box, Request tracing	Flow & Dependency	Statistical	Correlation Analysis, ANOVA
[32]	PAD	Cloud, System metrics	Black-box, Profiling	Knowledge-based	Machine Learning	Mutual Information, PCA, Semi-supervised Decision-tree
[33]	PAD	Distributed, Storage, Network request flows	Gray, Tracing	Flow & Dependency	Hybrid	C4.5, Regression Tree
[34]	PAD	Cloud, Web-based, System metrics	Black-box, Profiling	Observational	Hybrid	Markov-model, Tree Augmented Bayes
[35]	PAD	Cloud, System metrics	Black-box, Profiling	Knowledge-driven	Machine Learning	Supervised, One-class SVM
[17]	PADBI	Distributed, Storage, Application metrics	Black-box, Profiling	Knowledge-base	Statistical	SPC
[54]	PADBI	Cloud, Application & System metrics	Black-box, Profiling	Observational	Hybrid	Regression(LAR), Clustering
[62]	PADBI	Cloud, System metrics	Black-box, Profiling	Observational	Machine Learning	Self-Organizing Maps
[36]	PAD	Web-based, System metrics	Gray-box, Profiling	Observational	Machine Learning	Clustering, LOF
[60]	PADBI	Cloud	Black-box	Flow & Dependency	Statistical, Signal Processing	CUSUM, FFT Filtering
[37]	PAD	Cloud, System metrics	Black-box, Profiling	Observational	Machine Learning	Wavelet, Sliding Window
[26]	PBI	Multi-tier, Component & Resource bottlenecks	Blackbox	Observation	-	Fine-grained Load, Throughput Analysis
[47]	PAD	Distributed, , Resource metrics	Blackbox, Profiling	Observation	Statistical	Correlation Analysis
[39]	PAD	Cloud, Web-based, System metrics	Black-box, Profiling	Observational	Machine Learning	Hidden Markov Model, k-Nearest Neighbour, K-means Clustering
[40]	PAD	Cloud, System metrics	Black-box, Profiling	Knowledge-based	Machine Learning	LOF
[38]	PAD	Distributed, Hadoop Clusters, Component and Host bottlenecks	Black-box, Profiling	Observational	Machine Learning	Non-parametric Clustering
[41]	PAD	Web-based, System metrics	Gray-box, Profiling	Observation	Machine Learning	Clustering, LOF
[55]	PADBI	Cloud, System metrics	Black-box, System-call tracing	Observation	Statistical	Tukey Limits

ACM Copyright Form and Audio/Video Release

Title of the Work: Performance Anomaly Detection and Bottleneck Identification

Publication: Computing Surveys

Author/Presenter(s): Olumuyiwa Ibidunmoye (Umea University), Francisco Hernandez-Rodriguez (Umea University), Erik Elmroth (Umea University)

Auxiliary Materials (provide filenames and a description of auxiliary content, if any, for display in the ACM Digital Library. The description may be provided as a ReadMe file):

I. Copyright Transfer, Reserved Rights and Permitted Uses

* Your Copyright Transfer is conditional upon you agreeing to the terms set out below.

Copyright to the Work and to any supplemental files integral to the Work which are submitted with it for review and publication such as an extended proof, a PowerPoint outline, or appendices that may exceed a printed page limit, (including without limitation, the right to publish the Work in whole or in part in any and all forms of media, now or hereafter known) is hereby transferred to the ACM (for Government work, to the extent transferable) effective as of the date of this agreement, on the understanding that the Work has been accepted for publication by ACM.

Reserved Rights and Permitted Uses

(a) All rights and permissions the author has not granted to ACM are reserved to the Owner, including without limitation the ownership of the copyright of the Work and all other proprietary rights such as patent or trademark rights.

(b) Furthermore, notwithstanding the exclusive rights the Owner has granted to ACM, Owner shall have the right to do the following:

(i) Reuse any portion of the Work, without fee, in any future works written or edited by the Author, including books, lectures and presentations in any and all media.

(ii) Create a "Major Revision" which is wholly owned by the author

(iii) Post the Accepted Version of the Work on (1) the Author's home page, (2) the Owner's institutional repository, or (3) any repository legally mandated by an agency funding the research on which the Work is based.

(iv) Post an "Author-Izer" link enabling free downloads of the Version of Record in the ACM Digital Library on (1) the Author's home page or (2) the Owner's institutional repository;

(v) Prior to commencement of the ACM peer review process, post the version of the Work as submitted to ACM ("Submitted Version" or any earlier versions) to non-peer reviewed servers;

(vi) Make free distributions of the final published Version of Record internally to the Owner's employees, if applicable;

(vii) Make free distributions of the published Version of Record for Classroom and Personal Use;

(viii) Bundle the Work in any of Owner's software distributions; and

(ix) Use any Auxiliary Material independent from the Work.

When preparing your paper for submission using the ACM templates, you will need to include the rights management and bibstrip text blocks below to the lower left hand portion of the first page. As this text will provide rights information for your paper, please make sure that this text is displayed and positioned correctly when you submit your manuscript for publication.

Authors should understand that consistent with ACM's policy of encouraging dissemination of information, each work published by ACM appears with a copyright and the following notice:

> *Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from Permissions@acm.org.*
>
> *Copyright © 2015 0360-0300/2015/MonthOfPublication - ArticleNumber $15.00*

NOTE: DOIs will be registered and become active shortly after publication in the ACM Digital Library

☑ A. Assent to Assignment. I hereby represent and warrant that I am the sole owner (or authorized agent of the copyright owner(s)), with the exception of third party materials detailed in section III below. I have obtained permission for any third-party material included in the Work.

☐ B. Declaration for Government Work. I am an employee of the National Government of my country and my Government claims rights to this work, or it is not copyrightable (Government work is classified as Public Domain in U.S. only)

Country:

II. PERMISSION FOR CONFERENCE TAPING AND DISTRIBUTION (Check A and, if applicable, B)
A. Audio /Video Release

I hereby grant permission for ACM to include my name, likeness, presentation and comments in any and all forms, for the Conference and/or Publication.

I further grant permission for ACM to record and/or transcribe and reproduce my presentation as part of the ACM Digital Library, and to distribute the same for sale in complete or partial form as part of an ACM product on CD-ROM, DVD, webcast, USB device, streaming video or any other media format now or hereafter known.

I understand that my presentation will not be sold separately by itself as a stand-alone product without my direct consent. Accordingly, I give ACM the right to use my image, voice, pronouncements, likeness, and my name, and any biographical material submitted by me, in connection with the Conference and/or Publication, whether used in excerpts or in full, for distribution described above and for any associated advertising or exhibition.

Do you agree to the above Audio/Video Release? [✓ Yes] No

B. Auxiliary Materials, not integral to the Work

I hereby grant ACM permission to serve files named below containing my Auxiliary Material from the ACM Digital Library. I hereby represent and warrant that my Auxiliary Material contains no malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software, and I hereby agree to indemnify and hold harmless ACM from all liability, losses, damages, penalties, claims, actions, costs and expenses (including reasonable legal expense) arising from the use of such files.

☐ I agree to the above Auxiliary Materials permission statement

III. Third Party Materials

In the event that any materials used in my presentation or Auxiliary Materials contain the work of third-party individuals or organizations (including copyrighted music or movie excerpts or anything not owned by me), I understand that it is my responsibility to secure any necessary permissions and/or licenses for print and/or digital publication, and cite or attach them below.

[✓ We/I have not used third-party material.]
We/I have used third-party materials and have necessary permissions.

IV. Artistic Images

If your paper includes images that were created for any purpose other than this paper and to which you or your employer claim copyright, you must complete Part IV and be sure to include a notice of copyright with each such image in the paper.
[✓ We/I do not have any artistic images.]
We/I have any artistic images.

V. Representations, Warranties and Covenants

The undersigned hereby represents, warrants and covenants as follows:

(a) Owner is the sole owner or authorized agent of Owner(s) of the Work;

(b) The undersigned is authorized to enter into this Agreement and grant the rights included in this license to ACM;

(c) The Work is original and does not infringe the rights of any third party; all permissions for use of third-party materials consistent in scope and duration with the rights granted to ACM have been obtained, copies of such permissions have been provided to ACM, and the Work as submitted to ACM clearly and accurately indicates the credit to the proprietors of any such third-party materials (including any applicable copyright notice), or will be revised to indicate such credit;

(d) The Work has not been published except for informal postings on non-peer reviewed servers, and Owner covenants to use best efforts to place ACM DOI pointers on any such prior postings;

(e) The Auxiliary Materials, if any, contain no malicious code, virus, trojan horse or other software routines or hardware components designed to permit unauthorized access or to disable, erase or otherwise harm any computer systems or software; and

(f) The Artistic Images, if any, are clearly and accurately noted as such (including any applicable copyright notice) in the Submitted Version.

[✓] I agree to the Representations, Warranties and Covenants

DATE: **01/10/2015** sent to muyi.ibidun@gmail.com at **02:01:17**