CloudPD: Problem Determination and Diagnosis in Shared Dynamic Clouds

Bikash Sharma[1], Praveen Jayachandran[2], Akshat Verma[2], and Chita R. Das[1]

[1] Pennsylvania State University
{bikash,das}@cse.psu.edu
[2] IBM Research India
{praveen.j,akshatverma}@in.ibm.com

Technical Report CSE-12-002, May 2012

Abstract. Cloud computing has emerged as a popular computing paradigm allowing workloads to automatically scale in response to changes in demand. Clouds use virtualization to enable elasticity by continually reconfiguring the allocation of physical and virtual resources to workloads. Continual changes in clouds may lead to unexpected performance anomalies and traditional problem determination techniques are unable to deal with such cloud-induced anomalies. In this work, we study problem determination in virtualized clouds. We show that sharing of non-virtualized resources, frequent reconfiguration, higher propensity of faults, and automated steady state management in cloud pose new challenges for problem determination systems. We present the design and implementation of *CloudPD*, which uses a layered online learning approach to deal with frequent reconfiguration and high rate of faults, and is apathetic to the specific application(s) being executed. *CloudPD* models the operating context of a workload to capture the impact of sharing. It diagnoses anomalies using problem signatures and uses an end-to-end feedback loop that allows problem remediation to be integrated with cloud steady state management systems. We demonstrate that *CloudPD* achieves an accuracy of 88%, 83% and 83% for Hadoop, Olio and RUBiS workloads, respectively, with low false positives. In an end-to-end integrated case study with a cloud steady state management system running multiple applications, *CloudPD* achieved an accuracy of 77% and diagnosed errors in less than 30 seconds, clearly establishing its effectiveness in real-life operations.

1 Introduction

Large data centers and utility clouds experience frequent faults, which are top contributors of their management costs, and lead to SLA violations of the hosted services [6, 8, 22, 43]. Recently, a leading cloud provider, Amazon EC2, experienced a power outage that brought down their servers for seven hours, incurring severe financial penalties [27]. The public repository of failure traces [13] across diverse distributed systems like Skype, Microsoft, Planetlab, clearly demonstrates the prevalence and the need for taming the faults for the successful operation of the distributed systems. A recent survey [11] shows increasing customer reluctance and frustration to move to clouds due to poor performance. For instance, due to unexpected poor application performance, about 33% of the customers will shift to a different competitor cloud provider. Another recent study [5] on 3 years worth forum messages concerning the problems faced by end users of utility clouds show that virtualization related problems contribute to around 20% of the total problems experienced. These faults also manifest into poor performance of the hosted applications and their SLA violations [10, 20, 26, 34].

Problem determination of distributed applications has been well studied by researchers [2, 6]. Traditional problem determination is geared towards building a model of an application running without errors. When an application's current performance does not match the model of its normal execution state, an anomaly is detected. Application administrators are then alerted, who typically fix the anomaly manually [12, 14, 16, 24]. Clouds present an automated and dynamic data center model, which conflicts with the manual/semi-automatic process of problem determination. The trend of an increasing number of performance anomalies in clouds further exacerbates this problem.

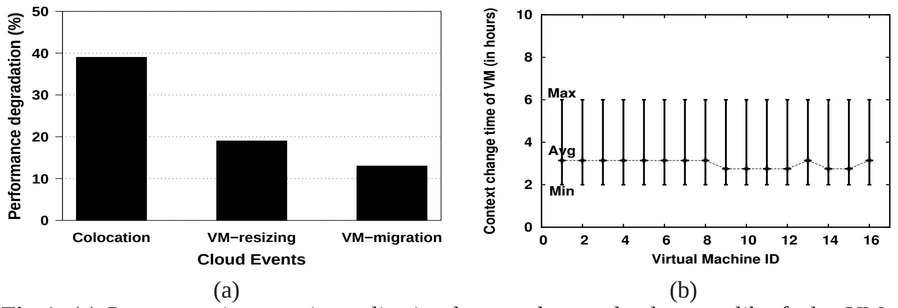


Fig. 1. (a) Percentage increase in application latency due to cloud events like faulty VM co-location, VM resizing and VM migration. (b) Rate at which a VM changes its operating context.

1.1 Problem Determination in Clouds: What is new?

In this work, we address the issue of problem determination in a multi-tenant dynamic IaaS (Infrastructure as a Service) cloud environment. We concern ourselves only with problems that lead to performance degradation, but do not cause the application to fail to execute altogether. In addition to scale, clouds present the following new challenges that traditional problem determination techniques fail to meet [5, 20, 21, 26, 34, 41].

- *Sharing of Resources*: Clouds are multi-tenant, and multiple virtual machines may be co-located on the same physical server. Since resources like caches and network bandwidth are not virtualized, the performance of an application may depend on other co-hosted applications. We note that multi-tenancy can lead up to a 40% performance degradation for a file system benchmark *Iozone* (Figure 1(a)). This makes it important for problem determination techniques to understand the *operating context* (the resource utilization behaviour of other co-located applications) under which a workload operates and distinguish a co-location fault from an application error. We define the notion of operating context in Section 3.
- *Dynamism*: Clouds are designed to be elastic and allow workloads to automatically request additional resources. Elasticity in a cloud is enabled using VM resizing, VM live migration, and VM cloning. Hence, the operating context under which a workload operates changes very frequently. In a case study (described in Section 5.6), we observed that the operating context of all VMs changes in 3.5 hours or less (Figure 1(b)), and the maximum duration without a change in operating context for any VM was 6 hours. This makes it imperative for a problem determination technique to learn the application behaviour in the specific operating context on

the fly. Hence, static model-based approaches do not work well in a cloud environment [20, 41, 42].

- *Higher Frequency of Faults*: Dynamism combined with sharing of resources leads to a large number of cloud induced anomalies. Cloud reconfiguration actions like VM resizing and VM live migration can lead to performance issues. We observed that a faulty VM resizing can impact performance by upto 30% and a faulty VM migration can impact migration by more than 10% (Figure 1(a)). Further, we observed that up to 10% of cloud reconfiguration actions can be faulty. Hence, problem determination in cloud needs to deal with a much higher frequency of faults than traditional distributed systems.
- *Autonomic Systems*: Cloud is an autonomic end-to-end system, which reacts automatically to changes in workload requirements [18]. A static problem determination system that flags a VM sizing error and waits for manual intervention does not fit the cloud model [21, 34, 42]. Problem determination in cloud needs to take a complete automated end-to-end approach of problem determination, diagnosis, classification and remediation by integrating with the cloud management stack.

1.2 Contribution

In this work, we present the design and implementation of the *CloudPD* system. *CloudPD* is designed for problem determination in multi-tenant elastic IaaS clouds and provides end-to-end problem determination and diagnosis. It uses an online learning approach to cope with frequent changes in the operating context of a workload. It considers the application as a black box and needs limited or no training to handle new applications. The following are the key contributions of our work:

- *CloudPD* can detect and diagnose both application related and cloud related anomalies with high accuracy and precision. Further, for cloud-related faults, *CloudPD* determines remediation actions and integrates with the cloud steady state management systems.
- To distinguish cloud related anomalies from application faults, *CloudPD* introduces the notion of an operating context for a workload. A change in application behavior due to change in operating context is not flagged as an application anomaly, if the application behavior is normal for the new operating context.
- In order to deal with the massive scale of cloud systems, frequent anomalies, and a large number of metrics monitored, *CloudPD* uses a 3-layered methodology with (i) a light-weight event generation phase, (ii) a moderately expensive correlation based problem determination phase for generated events, and (iii) a comprehensive problem diagnosis phase aided by pre-computed problem signatures followed by automated remediation, if applicable.
- We perform a comprehensive evaluation of *CloudPD* with cloud representative benchmarks – Hadoop, Olio, and Rubis, and a real enterprise traces-driven case-study on an IaaS cloud testbed, and show that *CloudPD* achieves high recall, precision, accuracy and low false positives. Further, it is able to suggest the required remediation action to the cloud resource manager in less than 30 seconds.

2 Background

We now present salient features of a cloud, which impact problem determination.

2.1 Continual Resource Optimization in Clouds

IaaS clouds support an on demand resource allocation model. In a cloud, a customer can sign up using a credit card and immediately access cloud resources to host virtual machines. Virtual machines can be automatically provisioned and deprovisioned with changing demand for a customer workload [3].

IaaS clouds, both public and private, use multi-tenancy to ensure that compute resources are utilized optimally. Since workloads can come in and leave dynamically, this leads to resource wastages. For example, a server hosting 5 VMs may be sub-optimally utilized if one or more VMs are removed or resized to use fewer resources. In order to ensure that a cloud always uses resources optimally, clouds leverage dynamic virtual machine consolidation [35]. Virtual machine consolidation uses VM resizing and VM live migration to ensure that all workloads are hosted on as few servers as possible at any given point in time. Hence, a cloud management system supports multi-tenancy and frequently employs one or more of VM cloning, VM resizing and VM live migration.

Multi-tenancy on a virtualized server is supported by reserving CPU and memory resources for individual virtual machines. However, virtualization does not allow reservation of resources that are not traditionally managed by operating systems. These include processor caches, memory bandwidth, and I/O bandwidth. Since all applications do not use these resources to the same extent, an application may see a change in performance if VMs either are added or removed from a physical server hosting the workload. The impact of cache contention [36], memory bandwidth [25] and network activity [9] has been well studied. Hence, clouds introduce co-location faults where an application experiences a performance anomaly due to a co-located VM.

VM resizing and VM live migration can also lead to performance anomalies for a workload. VM consolidation uses prediction to estimate the sizes of the VMs that host a workload [35]. An error in prediction may lead to a VM being allocated fewer resources. Hence, a faulty VM resizing may lead to a performance anomaly and needs to be detected, diagnosed, and appropriately remedied. VM live migration can similarly lead to performance degradation [19]. During VM live migration, all memory pages are marked as ready-only and writes lead to faults. Hence, the performance of write-intensive workloads can suffer during live migration. Further, live migration requires significant amount of CPU and this can lead to performance impact on live migrations as well as failed live migrations [18, 19, 38]. Hence, clouds introduce three new types of faults that need to be detected and distinguished from traditional application faults.

2.2 Problem Determination in a Cloud Ecosystem

An IaaS cloud management stack continually optimizes the cloud infrastructure. Cloud management systems are thus autonomic and automatically adapt to workload variations. Since future management actions in a cloud are dependent on the outcome of previous management actions, the cloud management stack should be immediately made aware of any faults that happen as a result of cloud management actions. Hence, a problem determination system in a cloud needs to be autonomic and provide fault remediation actions for cloud related faults.

Figure 2 captures the system context in which a cloud problem determination system needs to operate. A Cloud Resource Manager (CloudRM) periodically reconfigures the

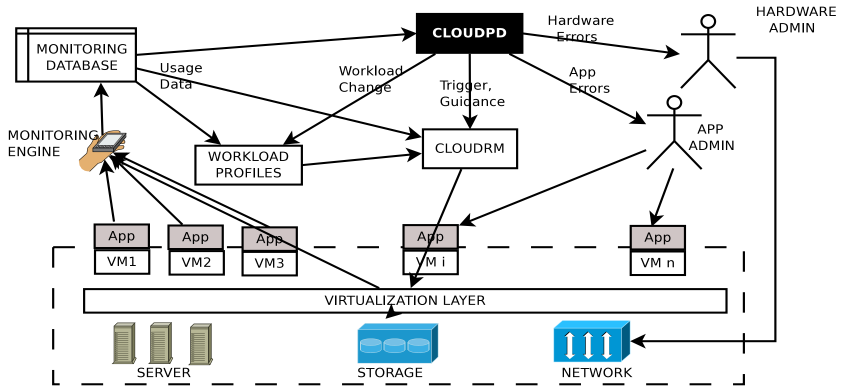


Fig. 2. System Context for *CloudPD* System

cloud using the monitored system and application data, and workload profiles as well. The monitored data is used to estimate the resource requirements and the workload profiles are used to identify workloads that may conflict with each other. The reconfiguration actions (VM resizing, VM cloning and VM live migration) are passed to the virtualization layer for action. A cloud problem determination system (*e.g., CloudPD*) needs to integrate with the cloud monitoring infrastructure to obtain live system and performance data as well as any other historical data it may need. Further, it needs to update any co-location errors as changes in workload profile. If any VM has been wrongly sized, *CloudPD* needs to trigger CloudRM to estimate the new sizes for the VM. Similarly, any faulty live migrations should be communicated as guidance to CloudRM for future configuration actions. Application and system errors are handled as usual by sending emails to application or system administrators.

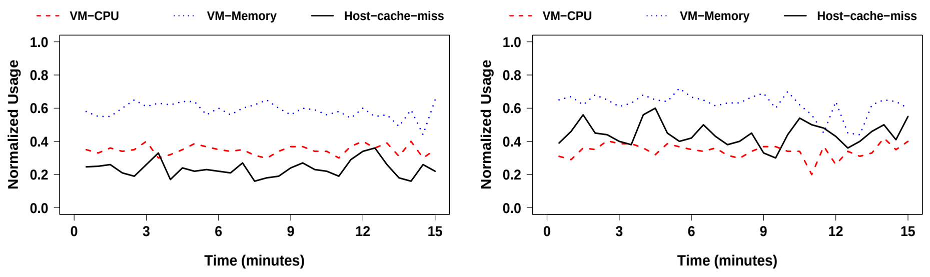
3 CloudPD Architecture

The design of *CloudPD* is motivated by the challenges for problem determination in clouds. We next present some of the key design decisions made by us to address them.

3.1 Design Decisions

CloudPD introduces the following ideas for practical problem determination in clouds.

– **Include Operating Context in Performance Model**: One of the fundamental design choices in *CloudPD* is to include the operating context of a workload in its performance model. Traditional problem determination techniques cover resource metrics that directly impact performance, such as CPU and memory, which are insufficient to differentiate application anomalies from cloud induced anomalies. We define an operating context for a VM to include (i) host metrics and (ii) impacted metrics. Impacted metrics are defined as those which are affected by the environment and include L1/L2 cache hits/misses, context switches, etc. Host metrics include the resource metrics (CPU, Memory usage) and impacted metrics for the physical server hosting the VM. The inclusion of operating context in the performance model allows *CloudPD* to differentiate environment changes from faults



(a) Normal interval (b) Anomalous interval

Fig. 3. Impact of cloud faults on operating context. The left and right plot, each represents a 15 minute normal and faulty interval, respectively. VM level parameters like CPU, memory remain stable, while the operating contexts like server cache miss experiences difference across.

inside the virtual machines. This is demonstrated in Figure 3. For both an anomalous VM migration interval and a normal interval, the CPU and memory utilization on a VM are nearly the same. However, the distinction that allows *CloudPD* to identify a migration fault is primarily in the cache misses experienced at the server hosting the VM.

– **Multi-layered approach**: The large intensity of faults in clouds coupled with inclusion of operating context makes problem determination very expensive for large scale clouds. Our second key design decision is to break problem determination into (i) a light-weight event generation phase (ii) a moderately expensive problem determination phase, and (iii) a comprehensive problem diagnosis phase.
– **Online Model Generation**: Frequent changes in operating context also imply that static model based approaches will not work well in clouds. Hence, *CloudPD* follows an online model generation approach for building performance models. This also allows *CloudPD* to consider the application as a black box and be easily applied to new applications.
– **Use of Peers to flag anomaly**: Clustered applications can benefit from comparing the monitored metrics across peers [20]. *CloudPD* also uses correlation of monitored metrics between peers to identify problems that affect single node.
– **Characterization of Anomalies using Correlations**: *CloudPD* uses pre-computed problem signatures of common anomalies. These problem signatures are expressed in terms of changes to correlations between VMs in a cluster for a metric, as well as correlations between metrics within a VM. We observed that these correlations are stable with changes in the number of VMs in a cluster, workload intensity and workload mix. Similar correlation-based techniques are also used in previous studies [8, 17, 20]. The problem signatures are used to uniquely perform detailed diagnosis of an anomaly allowing remediation to be automated.
– **Autonomic end-to-end problem detection and remediation**: *CloudPD* is designed as an autonomic system, which integrates with the cloud management systems. Hence, *CloudPD* includes an anomaly remediation manager that interacts with cloud steady state management systems to automatically suggest remediation actions to deal with cloud-related faults.

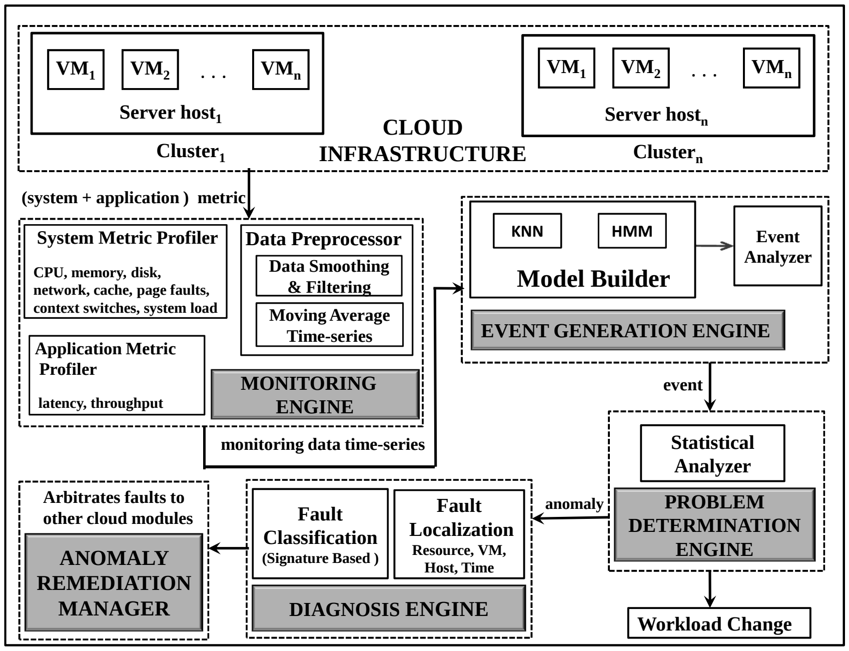


Fig. 4. Architecture of *CloudPD*.

3.2 Architecture

The architecture of *CloudPD* is shown in Figure 4. It comprises of five key components: (a) a Monitoring Engine that monitors each virtual machine and physical server for resource metrics, application metrics and the operating context; (b) an Event Generation Engine that quickly identifies a potential symptom of a fault and generates an event; (c) a Problem Determination Engine that further analyzes the event to determine deviations from normal behavior; (d) a Diagnosis Engine that classifies the anomaly based on expert knowledge; and (e) an Anomaly Remediation Engine that executes remedial actions on the diagnosed anomalies. In the following sub-sections, we describe each of these in detail.

Monitoring Engine The *Monitoring Engine* collects and processes various system metrics pertaining to CPU, memory, cache, network, and disk resources, and application metrics such as latency and throughput for every virtual machine and physical server. It is designed to capture (i) system resource metrics (ii) operating context and (iii) application performance metrics. A *System Metric Profiler* collects system resource metrics as well as the operating context, while *Application Metric Profiler* collects all the application performance metrics. All metrics are collected at periodic intervals with a configurable monitoring interval parameter. Operating systems and hypervisors perform a lot of background activity, which can introduce noise in the collected data. Hence, *CloudPD* has a *Data Preprocessor* module, which removes outliers. The *Data Preprocessor* takes the raw time-series data and generates *data-points*. A *data-point* is defined as a sequence of moving average values over a fixed interval of time and forms the basic input unit for our anomaly detection algorithms.

Event Generation Engine The *Event Generation Engine* implements the first phase of our multi-layered problem determination engine. It is designed to identify potential symptoms of anomalous behavior without performing a lot of computation. The generation of an event may not immediately denote the presence of a fault, but merely suggests the possibility of one. Further analysis would be needed to confirm if a fault is present, and if so, classify it.

In order for *Event Generation* to be light-weight, events are generated by looking at each monitored metric for each VM in isolation. Hence, an event only indicates that one or more performance metrics for a VM deviates from its performance model. A broader correlation between metrics across virtual machines can be very expensive as the number of such comparisons grow exponentially in $N \times M$, where N is the number of VMs and M is the number of monitored metrics for each VM. Note that the event generation engine can be parallelized with a separate thread performing the analysis for each (metric,VM) pair. Further, in order to build the performance model in an online fashion that is tolerant to workload variations, we use the nearest-neighbor algorithm, details of which are presented in Section 4.2.

Problem Determination Engine The Problem Determination Engine uses statistical correlation across VMs and resources to identify anomalies and localize them. For every event generated by the Event Generation Engine, this stage analyzes the data further to identify anomalies (the Problem Determination Engine is not invoked unless an event is generated). Let us suppose that an event was generated for metric M_j on VM VM_i. This stage then computes correlations between data for M_j and data for every other metric on VM_i, as well as correlations between data for M_j on VM_i and data for M_j on every other VM running the same application (wherever the information is available). The first set of correlations capture the relation between metrics for the VM (e.g., the correlation between resource metrics and their operating context), whereas the second set of correlations is based on the idea of using peer VMs to flag faults that occur in only one VM. Based on knowledge of typical correlation values under normal behavior, any significant deviations from normal correlation values are noted. If the deviations are larger than a threshold, a fault is generated and forwarded to the *Diagnosis Engine* for problem diagnosis.

In order to cope with dynamism, the *Problem Determination* phase computes models and the deviations in correlation described above in an online manner as shown in Algorithm 1. Correlations on data from an interval classified as normal from recent history is used to obtain a model of normal application behaviour. For the interval being analyzed, we compute similar correlations and check if these correlations deviate from the model of normal behaviour. The total number of correlations computed in this algorithm is of the order of the number of metrics plus the number of VMs running the application. Note that we do not analyze the parts of the system that are not affected, and only analyze the *neighborhood* of the location where an event is generated. This helps CloudPD to scale to large system sizes and several metrics monitored.

Diagnosis Engine The Diagnosis Engine uses pre-defined expert knowledge to classify the potentially anomalous scenarios detected by the Problem Determination Engine into one of several faulty and non-faulty classes. The expert knowledge is made available to the system in the form of standardized *fault signatures*. The fault signature captures

Algorithm 1 Correlation-based Problem Determination.

```
1:  Let $T_{normal-vm-r}$ = Data for VM $vm$ and metric $r$ over a normal interval from recent history;
    $T_{test-vm-r}$ = Data for VM $vm$ and metric $r$ combined over test interval and normal interval
2:  for each VM $i$ in cluster do
3:      if $ABS(corr(T_{test-vm-r}, T_{test-i-r}) - corr(T_{normal-vm-r}, T_{normal-i-r})) \geq Threshold$ then
4:          Flag deviation as anomaly for diagnosis.
5:      end if
6:  end for
7:  for each Metric $j$ do
8:      if $ABS(corr(T_{test-vm-r}, T_{test-vm-j}) - corr(T_{normal-vm-r}, T_{normal-vm-j})) \geq Threshold$
        then
9:          Flag deviation as anomaly for diagnosis.
10:     end if
11: end for
12: if No anomaly is flagged then
13:     Flag as normal workload change.
14: end if
```

the set of deviations from normal behavior, both in the correlation values computed by the Problem Determination Engine as well as in the operating environment, that are characteristic of the fault they describe. When anomalous behavior is detected, the deviations from normal behavior are matched with the known fault signatures. If a match is found, the Diagnosis Engine will successfully classify the fault. If not, it will flag it as an anomaly that it does not have knowledge of.

Anomaly Remediation Manager This component of CloudPD receives input from the Diagnosis Engine to perform suitable remedial actions. It is designed to deal with all the cloud related faults identified. In case of a co-location fault, *CloudPD* sends an exclusion rule to *CloudRM* that prevents co-location of the impacted VMs on a common physical server. In case of a faulty live migration, it provides new server utilization thresholds beyond which live migration should not be performed. In case of a VM sizing fault, it triggers resource estimation and resizing by *CloudRM*. For all other cases, a notification is sent to an application administrator or a system administrator.

3.3 Complexity Analysis

Let the number of VMs in a cluster executing the same application be N, the number of metrics monitored be M, and the number of fault types be F. Let T be the number of data points in each interval being analyzed. We make the following observations on the complexity of each of our stages of analysis:

- Event generation using kNN performs $O(NMT^2)$ work. This can be parallelized into $p \leq N$ threads, which would take $O(NMT^2/p)$ time.
- For each event generated, the problem determination stage performs $2(M+N)$ correlations (one each for the test interval data and the normal interval data obtained from recent history). Net complexity is $O((M+N)T^2)$.
- Let M' be the maximum number of deviations in correlations that are part of the fault signature used to uniquely identify each fault. This is typically a small number (< 10). For each anomaly identified, diagnosis takes $O(FM')$ time.

The layered approach to fault diagnosis combined with the low complexity of each of the stages, allows *CloudPD* to learn fault models in an online fashion and scale to large clusters of VMs.

4 Implementation Details

We implemented *CloudPD* to perform fault diagnosis for a VMWare ESX-based cloud cluster. We now provide details of our implementation.

4.1 Monitoring Engine

System Metrics	**Description**	**Measurement Level**
cpu-user	% CPU time in user-space	$VM, Host^*$
cpu-system	% CPU time in kernel-space	$VM, Host^*$
memused	% of used memory	$VM, Host^*$
miss/s	# of cache misses per second	$Host^*$
ctxt	context switches per second	VM^*
eth-rxbyt	network bytes received per second	VM
eth-txbyt	network bytes transmitted per second	VM
pgpgin	KBytes paged-in from disk per second	VM
pgpgout	KBytes paged-out from disk per second	VM
fault	page faults per second	VM
system-load	processor's process queue length	VM
Application Metrics	**Description**	**Measurement Level**
Latency	Response time	VM
Throughput	# of transactions/second	VM

Table 1. System and application metrics monitored in CloudPD; Metrics part of operating context are marked with *.

CloudPD's Monitoring Engine collects measurements from individual VMs as well as operating context parameters, for fault detection and diagnosis. The measurement data includes basic resource metrics (CPU, memory usage), impacted operating context parameters (e.g., context switches, cache hits/misses), host operating context parameters and application-level metrics (e.g, latency, throughput). Table 1 lists the set of system and application-level metrics monitored by CloudPD along with whether the metric is measured in the context of VM or at the physical host. We use Linux *sar* utility to collect the VM level system metrics. For getting server's cache hits/misses, we use VMware *vmkperf* [39] performance monitoring tool. To collect CPU and memory usage/reservation of the server, we use VMware *powercli cmdlets* [40]. The monitored data across all VMs and servers is collected via remote network copy, and stored at a central VM for further processing by subsequent stages.

4.2 Performance Models for Event Generation

We implemented three modeling techniques that can be used as part of the *Event Generation Engine*, namely, Hidden Markov Models (HMM), Nearest-neighbor, and k-means clustering [7]. All the three techniques attempt to qualitatively or quantitatively measure the extent of deviation between the current interval test data and past observations of normal behaviour. The CloudPD architecture allows plug and play of any modeling technique as part of the Event Generation Engine.

HMM-based Modeling This involves training a HMM to model the notion of normal behavior. As the application behavior may change with load or workload mix, to improve our accuracy we train multiple HMMs for different scenarios (wherever knowledge of typical application workload scenarios is available). As this step is intended to be fast and have a low overhead, we only create a limited number of such HMMs. A new test data point can be fed to each HMM to determine how well it fits the model. The advantage of this approach is that the HMM can capture the *sequence* of data samples and not just the set of samples that comprise the data point. The disadvantages of this technique are the need for training and that the model could be inaccurate for previously unobserved workload mixes. In our experience, we found that HMM was accurate in detecting faults that cause large deviations in observed values.

Nearest-Neighbor Techniques This technique works by computing a *distance* measure between the data point under consideration and the nearest *k* neighbors in a given set of model data points known to be normal from recent history. In our implementation, the distance between two data points was defined as the sum of the differences between corresponding samples in the two data points (other reasonable definitions of the distance metric worked equally well). The kNN distance measure for a data point is the sum of the distances to its *k* nearest neighbors. Larger the distance measure, the larger is the deviation from normal or expected behavior. If the distance measure for the test interval's data points are higher by a threshold compared to the distances of the model data points, an event (alarm) is generated (for further analysis by the problem determination engine). Note that, unlike HMM, this technique does not require any form of prior training and can learn the model of normal behaviour in an online fashion. Further, its online nature allows it to adapt to change in workload mix or intensity. Hence, *CloudPD* uses this technique as the default technique for event generation.

k-Means Clustering This is a well known statistical technique used for anomaly detection [7]. It works by clustering the data points, and if the test interval's data points form a different cluster compared to the interval from recent history known to be normal, then an alarm is raised. This technique is effective in detecting anomalies as identified by prior work in data center anomaly detection, but is NP-hard in complexity. This makes it unsuitable as a candidate for online model generation and analysis.

4.3 Diagnosis Engine

We adopt an XML format for describing the fault signatures, so as to allow the CloudPD system to learn to classify new kinds of faults with expert assistance. An assumption that we make here is that each fault has a characteristic signature and that the signature can be expressed in terms of the metrics monitored. We have observed this to be true in our extensive evaluations. CloudPD will be unable to detect faults which do not manifest as significant deviations in the metrics monitored, which is in fact a limitation of any fault detection and classification technique. We adopt a software wrapper based on

Matlab *xmlwrite* utility to implement the the diagnosis engine. Essentially, this wrapper provides two functionalities: (a) create a XML based signature of a new fault; (b) compare a newly created signature with existing fault signatures (stored in a database).

```
<event-list>
    <fault>
        <name=``CPU-hog-VM”></name>
        <category=``Software anomaly”></category>
        <description> A CPU-hog program is co-hosted on a VM </description>
        <signature>
            <VM-environment>
                <CPU-corr.diff> 0.20 </CPU-corr.diff>
                <system-load-corr.diff> 0.12 </system-load-corr.diff>
            </VM-environment>
            <Operating-environment>
              <Avg.CPU-diff(%)> 24.2 </Avg.CPU-diff(%)>
              <context-switches-corr.diff> 0.14 <context-switches-corr.diff>
            </Operating-environment>
            <Application-environment>
                <latency-diff(%)> 11.6 </latency-diff(%)>
            </Application-environment>
        </signature>
    </fault>
    <fault>
        <name=``VM under-sizing”></name>
        <category=``Invalid VM resource sizing”></category>
        <description> A VM is under sized to very low CPU and memory limits and ↩
             reservations </description>
        <signature>
            <VM-environment>
                <CPU-corr.diff> 0.13 <CPU-corr.diff>
                <memory-corr.diff> 0.11 </memory-corr.diff>
                <system-load-corr.diff> 0.09 </system-load-corr.diff>
            </VM-environment>
            <Operating-environment>
                <Avg.CPU-diff(%)> 15.6 </Avg.CPU-diff(%)>
                <Avg.memory-diff(%)> 9.2 </Avg.memory-diff(%)>
                <context-switches-corr.diff> 0.08 </context-switches-corr.diff>
            </Operating-environment>
            <Application-environment>
                <latency-diff(%)> 14.5 </latency-diff(%)>
            </Application-environment>
        </signature>
    </fault>
</event-list>
```

Fig. 5. Fault signature examples.

Figure 5 provides examples of expert-created signatures for two types of faults. The signatures are described using different contexts expressed as tags: (a) VM environment context captures the fault manifestation at the virtualized resources level; (b) operating environment context captures the metrics obtained at the physical host representing aggregate usage across all VMs residing on the host; (c) hypervisor context captures any special log messages obtained from the hypervisor and (d) application context captures application level performance metrics. A fault signature comprises of one or more of these context tags that characterize the fault it defines, and allows CloudPD to uniquely identify it. Only the metrics that deviate from normal behavior are captured in the signature, and are expressed as thresholds denoting the minimum deviation in the correlation values required for a match with the signature. For example, the CPU correlation value computed between a pair of VMs hosting an application should deviate from the correlation value under normal behavior by at least as large as the threshold defined as (*CPU-corr-diff*) to match the signature for wrong VM sizing (likewise, other correlation difference values also need to match).

4.4 Types of Faults Handled by *CloudPD*

Cloud anomalies	**Non-cloud anomalies**
Invalid VM resource sizing	Misconfigured application
Impact due to sharing	Workload mix change
Invalid VM live migration	Workload intensity change
VM reconfiguration	Software anomaly

Table 2. List of faults covered by *CloudPD*

Table 2 lists the various kinds of faults we detect in our *CloudPD* implementation. Details of each kind of fault and how they are simulated are given below:
Invalid VM resource sizing: In this fault, the resource allocation is misconfigured, where either CPU or memory reservation (or both) for a VM is very low leading to performance degradation of the application.
Impact due to co-location and resource sharing: This is a situation where two or more VMs hosted on the same physical server contend for a resource, and the performance of one or more VMs is affected. TODO: Refer to example in Section 1.
Invalid VM live migration: This type of fault reflects the problems that might arise due to live migration of virtual machines. We consider two scenarios. In the first scenario, we migrate a VM to a heavily congested physical host, where enough VMs are hosted on the server to saturate its CPU and memory capacity and the physical host has just enough capacity to accommodate the migrated VM. In the second scenario, we migrate a VM from a heavily congested source host, where there aren't sufficient CPU and memory resources to perform the migration.
VM reconfiguration error: This kind of fault arises during live VM migration, resizing, or during provisioning of new VM instances [37]. This happens when a VM is attempted to be configured in a way that conflicts with some configuration parameters of the physical host.
Misconfigured Application: This anomaly denotes the case where one of the application parameters has been set to an incorrect or invalid value. For Hadoop, there are around 190 MapReduce job configuration parameters [4]. To simulate this fault, we tweak some of these parameters like the number of mappers/reducers, DFS block size and DFS replication factor, and set them to invalid values. For Olio, we inject this fault by locking the *persons* table in the Olio database for the duration of the fault injection. Such misconfigurations are common in practice and can cause serious performance degradation [31, 33].

Workload mix change: This situation corresponds to changing the workload mix or the nature of the workload, which changes the extent to which different resources are utilized by the application. We use different browsing and bidding mix in Olio to implement this effect.
Workload intensity change: This indicates an increase or decrease in the intensity of the workload, while the nature of the workload itself remains the same. The motive here is to create scenarios, where the system is subjected to dynamic variations in workload. To simulate this for Hadoop, benchmarks like sort and distributed grep are run with different input sizes. For Olio, the load via Faban workload generator (in terms of the number of concurrent users) is varied dynamically.
Software Anomalies: Often software bugs manifest as performance anomalies that hog one or more resources. To simulate this in our experiments, we introduced programs that would hog resources.

CPU-hog: This represents a scenario where the application has gone into an infinite loop computation that is hogging the CPU. We inject this fault by introducing a piece of C code that performs floating point computations in an infinite loop, that runs on the target VM making it a faulty instance. This consumes most of the VM's available processor cycles, leading to poor application performance.

Memory-hog: This represents a scenario where there is a memory leak in the application. This is implemented by running a C program that consumes a lot of memory (it continuously reserves memory from heap using *malloc* without releasing the allocated chunks) on the target VM. The application is left with very little memory, leading to a significant drop in application throughput.

Disk-hog: This is similar to the CPU and memory hog scenarios. This is implemented using multiple Hadoop sort instances (on 5GB binary data each) running parallelly to create a high disk utilization scenario[3].

Network-hog: In this fault, the network link is heavily utilized. We implement this with a client VM using *httperf* [23] to establish a large number of persistent HTTP connections with a target VM, thus saturating its network.

Cache-hog In this fault, a cache-intensive benchmark from SPEC CPU2000 suite [32] is co-located with the target VM to simulate cache hog.

4.5 Anomaly Remediation Manager

The set of preventive or remedial actions taken by CloudPD steady state management system are summarized in Table 3.

5 Experimental Evaluation

We conducted extensive experiments to evaluate the performance of *CloudPD*. We first describe our experimental setup.

5.1 Experimental Setup

We used a virtualized server cluster to evaluate *CloudPD*. The server cluster consisted of 1 IBM x3850 M2 server with 16 Xeon 2.132 GHz cores and 132 GB RAM, 3 HS 21 bladeserver with 4 Xeon 2.33 GHz cores and 8 GB RAM each, and 1 HS 21 bladeserver

[3] refer Section 5 for experimental details

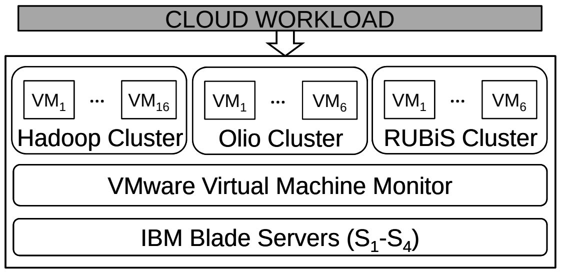


Fig. 6. Experimental testbed.

Fault	Catcher	Action
Wrong VM Resizing	CloudRM	Trigger re-configuration
VM to VM Contention	CloudRM	Create co-location exclusion constraints between the VM trigger reconfiguration
Invalid VM Migration	CloudRM	Update the migration cost for the VM (to be used in future consolidations)
Application Error	App Admin	Open a service ticket
System Error	CloudRM, App Admin	Update server list. Trigger reconfigurations for failover. Raise problem ticket
Workload Intensity Change	CloudRM	Same as Wrong VM resizing above
Workload Mix Change	CloudRM	Trigger re-profiling of the application. Reconfiguration, if needed by CloudRM

Table 3. Preventive actions taken by CloudPD+BrownMap at the onset of various kinds of faults.

with 4 Xeon 3 GHz cores and 10 GB RAM. The HS21 blades were hosted on an IBM Bladecenter-H chassis. The server have 2Gbps Fiber Channel port, which is connected to an IBM DS4800 Storage Controller via a Cisco MDS Fiber Channel Switch. All servers run the VMWare ESXi 4 Enterprise Plus hypervisor and are managed by a VMWare vSphere 4 server. The cloud setup supports enterprise virtualization features including live VM migration, live VM resizing and VM cloning.

We host 28 virtual machines on our cloud setup, which run Ubuntu server v10.04 64-bit operating system. Unless otherwise stated, all VMs are configured with 1.2 GB memory and 1.6 GHz CPU. Our virtual machines can be classified into three groups. The first group (*Hadoop Cluster*) consists of 16 VMs, and hosts a large scale distributed data processing framework called Hadoop MapReduce [15]. The second, *Olio Cluster* consists of 6 VMs (4 VMs in Web server tier and 2 VMs in database tier) and runs CloudStone [30], a multi-platform benchmark for Web 2.0 and cloud computing. CloudStone consists of a load injection framework called Faban, and a social online calendar Web application called Olio [30]. Faban is configured on a separate VM to drive the workload. The third (*RUBiS Cluster*) runs an E-commerce benchmark called RUBiS [29], and comprises of 6 VMs (2 VMs across each web server, application server and database tier, respectively). The outline of the testbed is shown in Figure 6.

5.2 Evaluation Metrics

We define the following four statistical measures to evaluate the effectiveness of anomaly detection and diagnosis by *CloudPD*.

$$Recall = \frac{\text{number of successful detections}}{\text{total number of anomalies}} \tag{1}$$

$$Precision = \frac{\text{number of successful detections}}{\text{total number of alarms}} \tag{2}$$

$$Accuracy = \frac{2 \times \text{Recall} \times \text{precision}}{\text{Recall} + \text{Precision}} \tag{3}$$

$$\textit{False Alarm Rate (FAR)} = \frac{\text{number of false alarms}}{\text{total number of alarms}} = 1 - Precision \tag{4}$$

5.3 Competitive Methodologies

To the best of our knowledge, there is no prior work that performs an end-to-end detection, diagnosis and classification of faults in virtualized cloud environments. Hence, for the purpose of evaluation, we extend some existing approaches to perform end-to-end problem determination in clouds. These schemes also implement a subset of our key design decisions helping us understand the importance of each of the design choice employed by *CloudPD*.
Baseline B1: This method is inspired from existing problem determination techniques that do not use operating context. Hence, $B1$ uses only VM CPU and VM memory usage for problem determination. This scheme employs all other *CloudPD* techniques including the three-layered approach, usage of peers, characterization of anomalies.
Baseline B2: This method does not use the multi-layer approach and analyzes every interval in detail for anomalies. Hence, $B2$ also acts as an oracle and defines the boundaries for any correlation based technique to solve problem determination in clouds.
Baseline B3: This method does not include the idea of correlation across peers. Hence, any gap between $B3$ and *CloudPD* can be attributed to the technique of correlating problem metrics across peers.
Baseline B4: This baseline uses static thresholds to trigger the execution of the Diagnosis Engine, contrary to *CloudPD*'s dynamically determined thresholds from the learnt models. Hence, comparison of this method with *CloudPD* can highlight the importance of using an online learning approach for problem determination in clouds.

5.4 Stability of Correlation with change in workload and configuration

We have used correlation across system metrics for a VM as well as correlation for a metric across peers to identify anomalies. Further, our problem signatures are also based on change in correlation between specific set of parameters. These problem signatures are defined independent of workload intensity, workload mix as well as VM configurations. It is an open question if correlation values remain stable with change in these parameters. Hence, our first set of experiments study if our assumption of correlation values being indepdent of these parameters is correct.

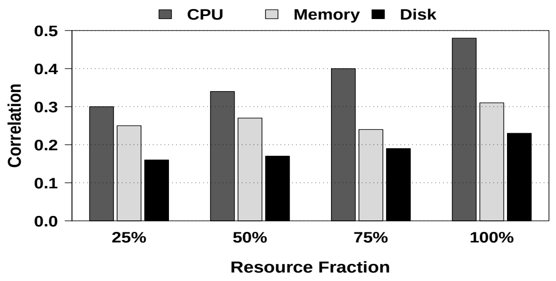
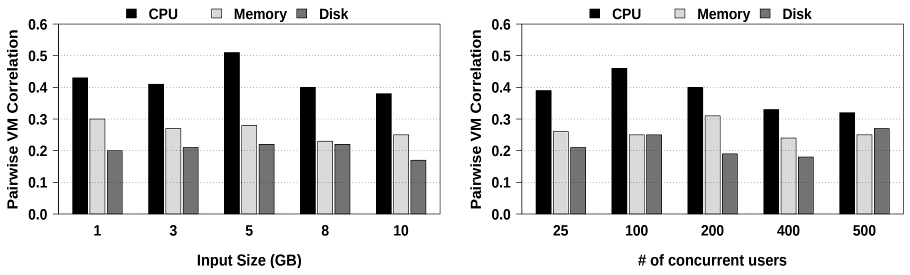


Fig. 7. Generality of *CloudPD* in homogeneous and heterogeneous environments.

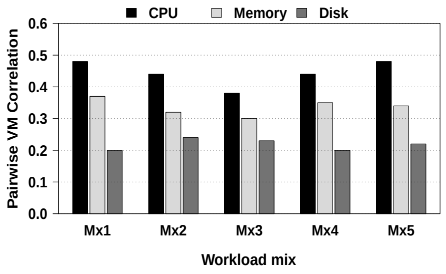


(a) Hadoop intensity (b) Olio intensity

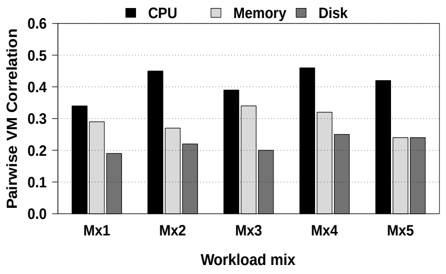
Fig. 8. Stability of peer (VM) correlation with change in workload intensity.

Stability of Correlation with Change in VM Configuration In this experiment, we vary the configuration of VMs running the workloads by varying their CPU and memory entitlements. We configured a cluster of 8 Hadoop VMs (VM1-VM8). The CPU and RAM reservations of VM1-VM4 are set to the default values of 1.2 GHz and 1.6 GB, respectively. Four experiments are conducted with different resource configurations for VM5-VM8 (25%, 50%, 75% and 100% of the CPU and memory of VM1-VM4). A Hadoop *Sort* benchmark is run on 5 GB of data on this Hadoop cluster. We correlate CPU and memory utilization across the VMs and report their average values in Figure 7. Although, the raw utilization might vary depending on the resource allocation to VMs, the correlation values remain nearly the same even for heterogeneous environments. This experiment establishes the stability of correlation values with change in VM configuration.

Stability of Correlation with Change in Workload Intensity We next demonstrate the stability of *CloudPD* with change in workload intensity changes. Figure 8(a) shows the variation in the average pairwise VM correlations (across CPU, memory and disk) with changes in workload intensity for Hadoop. The workload intensity change refers to running *Sort* with different input sizes (in the range of 1 to 10 GB). We can observe that normal workload change will not trigger a fault by *CloudPD*, since the pairwise VM correlations show very low variation across different workload intensities. As mentioned in Section 3.2, a fault affecting a specific VM will result in the correlation of the faulty VM with other VMs (for one or more metrics) to be quite deviant from other pairwise correlations. A similar observation holds good for the Olio benchmark as well as shown in Figure 8(b). For Olio, changes in workload intensity is achieved by changing the number of concurrent users from 25 to 500.

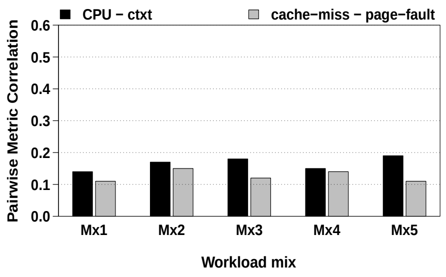


(a) Hadoop mix

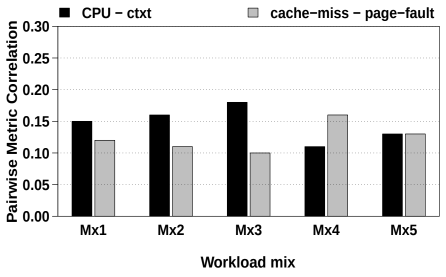


(b) Olio mix

Fig. 9. Stability of peer (VM) correlation with change in workload mix.



(a) Hadoop mix



(b) Olio mix

Fig. 10. Stability of metric correlation with change in workload mix. $CPU - ctxt$ refers to correlation between CPU and context switches on the same VM. $cache - fault$ implies correlation between cache misses and page faults.

Mix-type	Hadoop	Olio
Mx1	(streamSort,javaSort)-(s)	bm1+ 80% rw-bd
Mx2	(streamSort,combiner)-(l)	bm2 + 85% rw-bd
Mx3	(webdataScan,combiner)-(m)	bm3 + 90% rw-bd
Mx4	(combiner,monsterQuery)-(l)	bm4 + 95% rw-bd
Mx5	(webdataScan,webdataSort)-(s)	bm5 + 99% rw-bd

Table 4. Workload transaction mix specifications for Hadoop and Olio. For Olio, bm1-5 corresponds to different operation mixes in terms of browsing transactions.

Stability of Correlation with Change in Workload Mix The technique of correlating metric values across VMs running the same application works even with change in workload transaction mix. Changes in the transaction mix results in the resource usage behaviour of the application to change, but correlation across all the VMs remains stable with change in workload mix. The results of this experiment for Hadoop is shown in Figure 9(a), where we observe that the change in pairwise VM correlations is very low across different workload transaction mixes (the transaction mixes used are shown in Table 4). A similar observation is observed for Olio as well (Figure 9(b)). Furthermore, the correlation across metric pairs on the same VM also remain stable in case of normal workload transaction mix change, as depicted in Figure 10. Similar trend follows for same VM stable pairwise metric correlations in case of workload intensity change, but in the interest of space, we don't present the plots here.

5.5 Synthetic Fault Injection Results

Fault type	# of faults (Synthetic)	# of faults (Trace-driven)
invalid VM migration	3	4 (cloud induced)
invalid VM sizing	3	7 (cloud induced)
application misconfig.	4	0
CPU-hog	4	3
memory-hog	4	3
disk-hog	2	2
cache-hog	2	2
network-hog	2	2
Total faults	*24*	*23*

Table 5. Number of injected faults (Synthetic Injection) or induced faults (Trace-driven)

We compared the performance of various techniques in terms of their effectiveness to diagnose faults, diagnosis time, and scalability. In these experiments, we ran an application (Hadoop, Olio or RuBIS) for 24 hours. For Hadoop, we ran the *Sort* benchmark on 5 GB data repeatedly. For Olio, we used CloudStone [30] cloud computing benchmark. We used a setting of 100 concurrent users and the default Olio browsing transaction mix (*Mx*1 in Table 4). We use the EJB's stateless session beans implementation. We use the default workload mix for RUBiS (read-only browsing mix and bidding mix that includes 15% read-write interactions). We divided the experiment into 96 intervals of 15 minutes each. *CloudPD* collects data from the *Monitoring Engine* and performs a diagnosis for each interval using all the competing methods. We inject faults randomly in 24 of these 96 intervals. The details of the faults injected are presented in Table 5. An interval is categorized as anomalous if the application latency or throughput is above a certain threshold (obtained through empirical analysis) from the latency/throughput observed for normal behavior. For Hadoop, the application latency is the end-to-end job completion time; for RUBiS, latency is the end-to-end transaction response time. The latency threshold value chosen was 11%. For Olio, the application throughput is the number of transactions completed per second, and the threshold used was 9%.

End-to-end Diagnosis Comparison Table 6 shows the end-to-end diagnosis results for *CloudPD* and the competing methods. For Hadoop, *CloudPD* was able to correctly detect and diagnose 21 out of the 24 faults and 69 out of the 72 normal intervals. It compares very favorably with Oracle B2, which is able to identify only one more additional anomaly compared to *CloudPD* with exhaustive analysis. Baseline B1 has low recall and precision as it monitors only a VM's CPU and memory, and ignores other system metrics (both at VM and server level). Further, it also has a high false alarm rate, where change in operating context is classified as an anomaly. *B*1 wrongly diagnoses errors in few cases for the same reason. *CloudPD* is able to avoid this false alarms as it correlates data across multiple metrics (eg., CPU with context switches and memory with page faults) and does not report an anomaly if the correlations are consistent with the learnt models. Baseline B3 also recorded a low recall and precision and a high false alarm rate, as it does not correlate across peers (VMs running the same application).

Method	# of correct normal detections	# of correct anomalous detections	# of correct Phase 1	# of total predicted anomalies	Recall	Precision	Accuracy	FAR
CloudPD	69	21	23	24	0.88	0.88	0.88	0.12
B1	62	11	15	21	0.46	0.52	0.49	0.48
B2	69	22	24	25	0.92	0.88	0.90	0.12
B3	64	12	23	20	0.50	0.60	0.54	0.40
B4	66	15	15	21	0.63	0.71	0.67	0.29
Hadoop								
CloudPD	68	20	22	24	0.83	0.83	0.83	0.17
B1	62	11	15	21	0.46	0.52	0.49	0.48
B2	68	22	24	26	0.92	0.85	0.88	0.15
B3	63	11	22	20	0.46	0.55	0.50	0.45
B4	63	13	14	22	0.54	0.59	0.56	0.41
Olio								
CloudPD	68	20	22	24	0.83	0.83	0.83	0.17
B1	61	12	15	23	0.50	0.52	0.51	0.48
B2	68	22	24	26	0.92	0.85	0.88	0.15
B3	62	12	22	22	0.50	0.54	0.52	0.46
B4	63	14	13	23	0.59	0.61	0.60	0.39
RUBiS								

Table 6. Comparing End-to-end Diagnosis Effectiveness

Method	Undetected anomalies
CloudPD	1 disk hog + 2 application misconfig. (total 3)
B1	2 network hog + 2 disk hog + 2 cache hog + 2 application misconfig. + 2 invalid VM sizing + 3 invalid VM migration (total 13)
B2	2 application misconfig. (total 2)
B3	2 disk hog + 2 cache hog + 2 application misconfig. + 2 memory hog + 1 CPU hog + 3 invalid VM migration (total 12)
B4	2 disk hog + 1 cache hog + 1 memory hog + 2 application misconfig. + 3 invalid VM migration (total 9)

Table 7. Undetected anomalies for Hadoop.

However, its performance is better than *B*1. Baseline B4, that uses static thresholds again does not have satisfactory performance. Similar results are observed for Olio and RuBIS, although the performance of *CloudPD* is slightly poorer compared to Hadoop. We conjecture that the reason for this is because Hadoop is a symmetric application (map/reduce tasks for Hadoop are similar in type and the amount of work across all the VMs), whereas Olio is a more generic distributed application. There exists correlations among VMs in the same tier as well as correlations among VMs across the tiers for Olio. However, the magnitude of intra-tier VM correlations is higher than inter-tier VMs correlations, which results in some error in detecting anomalies.

We further analyze the specific faults that were undetected by *CloudPD* and the four baselines. Tables 7 and 8 list the anomalies that were missed by *CloudPD* and the baselines B1-B4, for Hadoop and Olio, respectively. *CloudPD* failed to identify 1 disk hog

Method	**Undetected anomalies**
CloudPD	2 disk hog + 2 application misconfig. (total 4)
B1	2 network hog + 2 disk hog + 2 cache hog + 2 application misconfig. + 2 invalid VM sizing + 3 invalid VM migration (total 13)
B2	2 application misconfig. (total 2)
B3	2 disk hog + 2 cache hog + 3 application misconfig. + 2 memory hog + 1 CPU hog + 3 invalid VM migration (total 13)
B4	2 disk hog + 1 cache hog + 2 memory hog + 3 application misconfig. + 3 invalid VM migration (total 11)

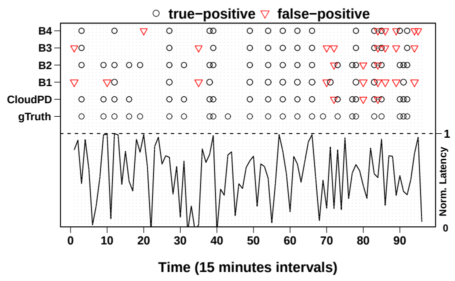
Table 8. Undetected anomalies for Olio.

Method	**Undetected anomalies**
CloudPD	2 disk hog + 2 application misconfig. (total 4)
B1	2 network hog + 2 disk hog + 1 cache hog + 2 application misconfig. + 2 invalid VM sizing + 3 invalid VM migration (total 12)
B2	2 application misconfig. (total 2)
B3	2 disk hog + 1 cache hog + 3 application misconfig. + 2 memory hog + 1 CPU hog + 3 invalid VM migration (total 12)
B4	2 disk hog + 1 cache hog + 2 memory hog + 3 application misconfig. + 3 invalid VM migration (total 10)

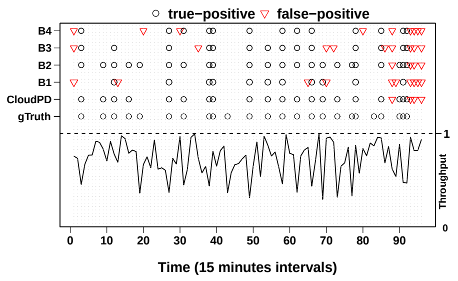
Table 9. Undetected anomalies for RUBiS.

and 2 application misconfiguration faults. The reason disk hog eluded detection can be explained by the fact that the VMs share their local disks across a Storage Area Network (SAN). The deviation of disk utilization from normal values was not sufficient for the event generation engine to raise an alarm, preventing *CloudPD* from detecting it. However, baseline B2 was able to detect this as correlations across metrics and VMs calculated by the problem determination engine showed a deviation from normal behaviour. *CloudPD* could not identify 2 application misconfiguration faults as well, as the difference in cross-resource and cross-VM correlation values from normal was not high enough to mark them as anomalies. B2 missed the same 2 application misconfiguration faults as *CloudPD*. As baseline B1 only monitors CPU and memory, it missed a total of 13 faults (it was effective in identifying only CPU and memory hog faults). Baseline B3 failed to detect most application related faults as it only performs correlations across resources within a VM and does not perform cross-VM correlations. A faulty VM considered by itself appears as though it is servicing a very large workload, and hence was not tagged as anomalous. Baseline B4 is sensitive to the specific thresholds that are set, and is ineffective in identifying different manifestations of the same type of fault.

Diagnosis Time and Scalability Effective diagnosis is just one of the goals for problem determination in cloud. A dynamic and autonomic system like cloud requires diagnosis to be performed quickly so that remediation actions can be taken. We next study the calibration and analysis time for each stage of analysis in Tables 10, 11 and 12. The numbers are averaged across the 96 intervals in the 24-hour experiment. We remind the reader that the system state can change every 15 minutes and hence remediation is relevant only if performed in time much less than 15 minutes. The event generation



(a) Hadoop



(b) Olio

Fig. 11. Behavior of system in the presence of faults.

Method	Event Generation	Problem Determination	Diagnosis Engine	Total
CloudPD	17.8	11.1	1.4	30.3
B1	13.2	8.9	1.3	23.4
B2	0	133	1.4	134.4
B3	17.7	6.3	1.3	25.3
B4	4.4	34	1.2	39.6

Table 10. Calibration and analysis time for Hadoop (units in seconds).

Method	Event Generation	Problem Determination	Diagnosis Engine	Total
CloudPD	11.3	7.1	1.4	19.8
B1	9.1	6.0	1.2	16.3
B2	0	90.6	1.4	92
B3	11.5	5.1	1.3	17.9
B4	3.8	19.5	1.2	24.5

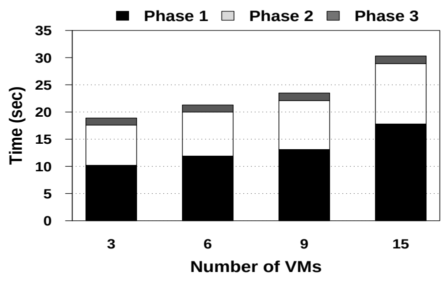
Table 11. Calibration and analysis time for Olio (units in seconds).

Method	Event Generation	Problem Determination	Diagnosis Engine	Total
CloudPD	11.1	7.1	1.3	19.5
B1	9	6.2	1.4	16.6
B2	0	91.3	1.3	92.6
B3	11.4	5.4	1.2	18
B4	3.9	19.7	1.2	24.8

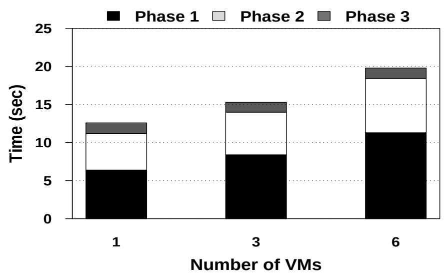
Table 12. Calibration and analysis time for RUBiS (units in seconds).

stage of *CloudPD* takes an average 17.8 seconds for Hadoop and is executed for every interval, and on data for every VM and metric monitored. The problem determination stage is triggered only if an alarm is raised by the event generation stage. For other intervals, the time taken by the problem determination stage is taken as zero while computing the average. Hence, although the problem determination stage takes longer than the event generation stage each time its invoked, as it is invoked only selectively, the average time spent at this stage is lower. The same is true for the diagnosis stage as this is invoked only if an anomaly is detected. Oracle B2 has no event generation phase and hence the time spent at problem determination is 10X larger than *CloudPD*. Baselines B1 and B3 have marginally lower analysis times than *CloudPD*, as they detect fewer anomalies and the expensive problem determination and diagnosis phases are not triggered as often. The time spent analyzing the Olio cluster is lesser compared to Hadoop as the cluster size is smaller. But, the observations made above hold true for Olio as well. One may also note that *CloudPD* spends most of the time in event generation which is completely parallelizable and can be easily reduced further.

We next study the scalability of *CloudPD* with increase in the number of VMs. Figure 12 shows the effect of increase in the application size (number of VMs hosting the application) on the calibration and analysis time of *CloudPD*. The analysis time is



(a) Hadoop VM scaling



(b) Olio VM scaling

Fig. 12. Effect of VM scaling on calibration and analysis time in *CloudPD*

shown as histograms with breakup of time taken by each stage of analysis for different number of VMs in the Hadoop and Olio clusters. We notice that the analysis time grows sub-linearly with the number of VMs, with the increase in analysis time being mainly in the first event generation phase. Further, as we have noted before, event generation time can be reduced by parallelizing event generation for different VMs. This experiment establishes the effectiveness of *CloudPD*'s multi-layer approach, allowing it to scale to larger systems with more VMs and servers.

Workload	HMM	KNN	KMeans
Hadoop	8.9	4.3	6.2
Olio	9.2	4.4	6.4

Table 13. Training time for Hadoop and Olio (units in seconds).

We next present the time taken to train the models for normal and anomalous behaviour using three different techniques for the event generation phase of *CloudPD*, namely HMM, kNN, and k-Means. The numbers are averaged across several runs (total 5), which aided the formulation of the fault signatures outlined in Section 3.2. We note that *CloudPD* uses very limited training, which allows it to learn in an online fashion from experience. It is able to adapt to changes in application behaviour (demonstrated in the case study experiment in Section 5.6) and can learn to identify new anomalies.

5.6 Trace-driven Fault Generation Results

Our next set of experiments were conducted to study the effectiveness of *CloudPD* in a real cloud setting, where cloud configuration actions automatically generate faults. Our cloud testbed is managed by a cloud management stack that could provision virtual machines and perform dynamic consolidation to reduce power. We used the $pMapper$ consolidation manager [35], which has been studied by multiple researchers and productized. The consolidation involves reconfiguration of VM resource allocations as well as changing the placement of VMs on physical servers every 2 hours. The cloud testbed hosts 16 VMs, which ran Hadoop and Olio. The Hadoop cluster consisted of 10 VMs and the Olio cluster consisted of 6 VMs, with 4 VMs for the web server tier, and 2 VMs for the database tier.

We used two real traces from a production data center of a Fortune 500 company running enterprise applications to drive the workload for Hadoop and Olio. The two traces contain a time-series of CPU and memory utilization across the servers, with a data point every 15 minutes for a total duration of 24 hours, which is also the duration of our experiment. We built a profile of *Hadoop* that captured the relationship of workload size with CPU and memory utilization. We similarly created an application profile of *Olio* to capture the relationship of number of concurrent users with CPU and memory utilization of the Web and database tier. In a given interval, we ran Hadoop with a data size that generates the CPU utilization provided by the trace for that interval. Similarly, we ran Olio with number of users such that the utilization of the web server matched the CPU utilization specified by the trace for the interval. Hence, this experiment captures changes in workload intensity that happen in real clouds. The resource profiles are captured by the following equation.

$$CPU(Olio-Web) = (Users/10+5), CPU(Olio-DB) = Users*0.035+7.5 \quad (5)$$

$$CPU(Hadoop) = DataSize*2.83+12.9 \quad (6)$$

CloudPD independently monitors the cluster and identifies cloud related anomalies, workload intensity and workload mix changes, as well as application anomalies for 15 minute intervals. We randomly inject anomalies in some of these intervals as listed in Table 5. Apart from the injected application anomalies, we noticed that 4 intervals experienced invalid VM migrations and 7 intervals experienced invalid VM sizing anomalies due to cloud reconfiguration (we do not inject any cloud related anomalies). These were determined to be anomalous as the application latency and throughput exceeded 11% and 9% for Hadoop and Olio, respectively. The sizing faults were a result of prediction error and the live migration faults were due to high resource utilization on the servers.

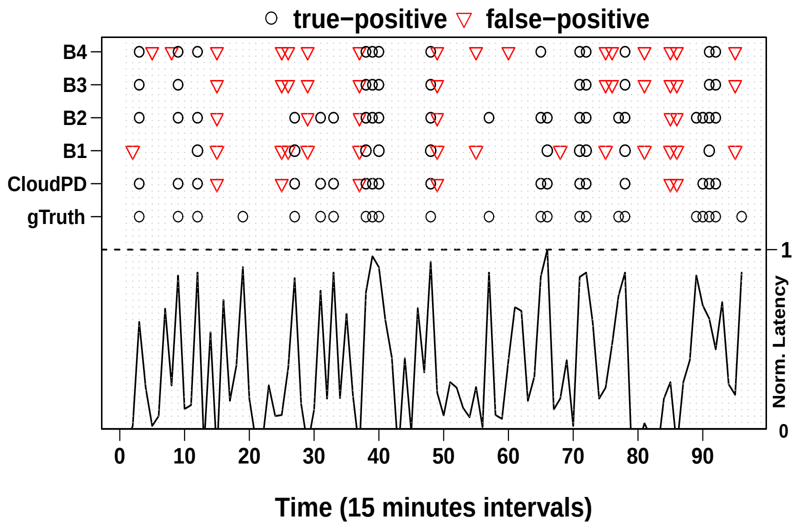


Fig. 13. Fault Diagnosis with Time

Diagnosis Effectiveness Figure 13 provides a detailed view of the case-study. For each of the 96 intervals (shown in the x-axis), the left side y-axis shows the ground truth of

Method	# of correct normal detections	# of correct anomalous detections	# of correct Phase1	# of total predicted anomalies	Recall	Precision	Accuracy	FAR
CloudPD	67	18	21	24	0.78	0.75	0.77	0.25
B1	58	10	14	25	0.43	0.40	0.42	0.60
B2	67	21	23	27	0.91	0.78	0.84	0.22
B3	60	11	21	24	0.48	0.46	0.47	0.54
B4	60	13	15	26	0.57	0.50	0.53	0.50

Table 14. Comparing end-to-end diagnosis effectiveness for trace-driven study

intervals that had anomalies along with anomaly predictions made by *CloudPD* and the four baselines. Both correct anomalous detections and false positives are marked. On the y2-axis, we plot the normalized application latency, where a value of 1 denotes the interval with the highest average job latency. Observe that anomalous intervals (marked as $gTruth$) have a higher application latency than normal intervals. One can visually observe that $B1$, $B3$ and $B4$ have a large number of false positives, which is consistent with our studies using synthetic fault injection (Section 5.5).

The performance metrics for *CloudPD* and the baselines in terms of recall, precision, and other metrics are summarized in Table 14. *CloudPD* is able to correctly diagnose 18 out of the 23 anomalous intervals and identify 67 out of the 73 normal intervals as normal. This is close to the performance of baseline $B2$ and significantly better than the other three baselines. Table 15 provides the list of anomalies that were undetected by *CloudPD* and the baselines. *CloudPD* fails to detect the 2 disk hog anomalies, the reason for which can be attributed to the fact that the VMs share a SAN storage and the deviation of the storage utilization values from normal was not significant enough for the event generation phase of *CloudPD* to raise an alarm. *CloudPD* also missed detecting 2 invalid resizing events and an invalid VM migration. These were marginal anomalies which caused the application latency to be high enough to be classified as an anomaly, but the correlation values were not deviant enough for *CloudPD* to detect it.

Method	Undetected anomalies
CloudPD	2 disk hog + 2 invalid VM sizing + 1 invalid VM migration (total 5)
B1	2 network hog + 2 disk hog + 2 cache hog + 3 invalid VM sizing + 4 invalid VM migration (total 13)
B2	1 disk hog + 1 invalid VM sizing (total 2)
B3	2 disk hog + 2 cache hog + 2 memory hog + 1 CPU hog + 2 invalid VM sizing + 3 invalid VM migration (total 12)
B4	2 disk hog + 2 cache hog + 1 memory hog + 1 CPU hog + 2 invalid VM sizing + 2 invalid VM migration (total 10)

Table 15. Undetected anomalies for case study.

Method	**Event Generation**	**Problem Determination**	**Diagnosis Engine**	**Total**
CloudPD	18.5	11.9	1.3	31.7
B1	14.1	9.1	1.4	24.6
B2	0	135.5	1.4	136.9
B3	18.5	5.9	1.3	25.7
B4	5	36.7	1.3	43

Table 16. Calibration and analysis time for case-study (units in seconds).

Diagnosis Time The calibration and analysis time for *CloudPD* and the baselines for each stage of analysis is summarized in Table 16. If the problem determination and diagnosis stage was not triggered for a particular interval, the time taken for that interval was taken as zero while computing the average. We note that *CloudPD* has a low analysis time allowing it operate in an online fashion in a realistic cloud environment and can scale to large data centers. The analysis time is significantly lower than that of baseline B2, demonstrating the efficiency of the event generation phase of the analysis.

5.7 Diagnosis Overheads

	% CPU	Memory (MB)	Network BW (KB/s)
Hadoop	0.35	0.80	37.8
Olio	0.18	0.44	14.5
Case study	0.41	0.92	40

Table 17. CPU usage (% CPU time), memory usage (MB) and network bandwidth (BW) overhead during the data collection using *sar* utility in Linux.

The problem determination framework uses cloud resources for performing the diagnosis of faults. We also observed the overhead on managed virtual machines due to *CloudPD*. We quantify the overhead of *CloudPD* in terms of the CPU, memory and network bandwidth usage for both our experiments with synthetic faults as well as the trace-driven study. We report the resource utilization averaged across VMs and over the duration of entire experiment (*i.e.*, 24 hours) in Table 5.7. We observe that *CloudPD* introduces minimal overhead on the system and is able to detect anomalies with high accuracy and low false positives. Our experimental study thus establishes the effectivness of *CloudPD* to accurately diagnose application, system, and cloud-induced faults quickly, and with low overhead.

6 Related Work

Problem determination in distributed systems in general is an important system management artifact and has been thoroughly studied in prior literature. Some of these studies can be classified into the following based on the type of techniques used:
(a) *Threshold-based schemes:* In this approach, an upper/lower bound is set for each system metric being monitored. These thresholds are determined based on the historical data analysis or predefined application-level performance constraints like QoS or SLAs.

On violation of the threshold for any metric being monitored, an anomaly alarm is triggered. This methodology forms the core of many of the commercial [12, 16] and some open source [14, 24] monitoring tools. This methodology however suffers from high false alarm rate, static and off-line characteristics, and is expected to perform poorly in the context of large scale utility clouds [42].
(b) *Statistical machine learning techniques:* Many anomaly detection schemes leverage machine learning and statistical methods. Anomaly detection tools like like E2EProf [1] and Pinpoint [8] use various statistical techniques like correlation, clustering, entropy, profiling and analytical modeling, for the identification of performance related problems in distributed systems.

Problem determination in clouds: Existing problem detection schemes and tools for traditional distributed systems would fail to address the challenges of fault management in virtualized cloud systems for the reasons outlined in Section 1. This has brought enough motivations that recently researchers have started exploring fault management issues in virtualized systems like clouds.Kang et al. [20] proposed PeerWatch, a fault detection and diagnosis tool for virtualized consolidated systems. PeerWatch utilizes a statistical technique, cannonical correlation analysis to model the correlation between multiple application instances to detect and localize faults. EbAT [42] is a system for anomaly identification in data centers. EbAT analyzes system metric distributions rather than individual metric thresholds. Vigilant [28] is an out-of-band, hypervisor-based failure monitoring scheme for virtual machines, that uses machine learning to identify faults in the VMs and guest OS. DAPA [21] is an initial prototype of application performance diagnostic framework for a virtualized environment. PREPARE [34] is a recently proposed framework for performance anomaly prevention in virtualized clouds. PREPARE integrates online anomaly prediction and predictive prevention to minimize the performance anomaly penalty. The main focus of all these initial works in the context of virtualized environment is to diagnosis application performance anomalies and identify causes of SLA violations. A recent study [5] on 3 years worth forum messages concerning the problems faced by end users of utility clouds show that virtualization related problems contribute to around 20% of the total problems experienced.

The above mentioned prior works both in the context of traditional distributed systems and utility clouds have only addressed software bug or application related faults, which manifest into performance anomalies. However, none of these have focused on cloud-centric faults that arise from frequent, dynamic reconfiguration activities in clouds like wrong VM sizing, VM live migration, and anomalies due to collocation, *i.e.*, anomalies that arise due to sharing of resources across VMs consolidated on the same physical hardware. We make an attempt to address these type of faults in particular and the associated challenges in their diagnosis in this paper.

The other important dimensions where our proposed fault management framework, *CloudPD*, differ from these related works include: (i) we monitor and analyze a diverse and complete list of system metrics (see Table 1) to better capture the system context, compared to only CPU, memory metrics being considered in most prior works; (ii) most prior literature talk about efficient anomaly detection, but very few go beyond that and address diagnosis of these faults after detection. The Diagnosis Engine of *CloudPD* localizes the root cause of fault in terms of affected system metrics, VM, server, and application component. Further, it also classifies these anomalies into known fault type for better handling of similar future anomalies. Moreover, the Anomaly Remediation Manager handles preventive and remedial actions by coordinating with other cloud modules. These pieces are absent in previous works; (iii) a narrow range of application benchmarks are considered for evaluations in previous works. We evaluate *CloudPD* with three representative cloud workloads – Hadoop, Olio and RUBiS, and also a case study with real traces from an enterprise application; (iv) small experimental test-bed has been used in previous studies (5-8 VMs). We have used a comparatively larger evaluation system consisting of 28 virtual machines, consolidated on 4 blade servers. Due to the aforesaid reasons, we to the best of our knowledge believe to be the first to address fault/anomaly detection and localization for cloud-centric virtualized infrastructure, and covering a larger set of applications/workloads and experimental test-bed.

7 Conclusions and Future Work

In this paper, we proposed a light-weight, automated, and accurate fault detection and diagnosis system called *CloudPD* for shared utility clouds. *CloudPD* uses a layered online learning approach to deal with the frequent reconfiguration and higher propensity of faults in clouds. We introduced the notion of operating context, which is essential to identify faults that arise due to the shared nature of non-virtualized resources. *CloudPD* diagnoses anomalies based on pre-computed fault signatures and allows remediation to be integrated with cloud steady state management in an automated fashion. We demonstrated using extensive experimentation that *CloudPD* achieves high accuracy and precision in detecting and distinguishing cloud-related faults from application faults and workload changes, within tens of seconds.

As part of future work, we would also like to explore CloudPD with more diverse cloud benchmarks and test its effectiveness on a large commercial cloud testbed like IBMSmartCloud. We plan to extend CloudPD with more fault scenarios typical to virtualized environment, coupled with an extensive fault signatures database.

References

1. S. Agarwala, F. Alegre, K. Schwan, and J. Mehalingham. E2EProf: Automated End-to-End Performance Management for Enterprise Systems. In *IEEE DSN*, 2007.
2. S. Agarwala and K. Schwan. SysProf: Online Distributed Behavior Diagnosis through Fine-grain System Monitoring. In *ICDCS*, 2006.
3. Amazon. Auto scaling. `http://aws.amazon.com/autoscaling`.
4. Shivnath Babu. Towards Automatic Optimization of MapReduce Programs. In *ACM SOCC*, 2010.
5. T. Benson, S. Sahu, A. Akella, and A. Shaikh. A First Look at Problems in the Cloud. In *USENIX HotCloud*, 2010.
6. P. Bodik, M. Goldszmidt, A. Fox, D. Woodard, and H. Andersen. Fingerprinting the Datacenter: Automated Classification of Performance Crises. In *EuroSys*, 2010.
7. V. Chandola, A. Banerjee, and V. Kumar. Anomaly detection: A survey. *ACM Computing Surveys*, 41, 2009.
8. M.Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: Problem Determination in Large, Dynamic Internet Services. In *IEEE DSN*, 2002.
9. L. Cherkasova and R. Gardner. Measuring CPU Overhead for I/O Processing in the Xen Virtual Machine Monitor. In *Usenix ATC*, 2005.

10. L. Cherkasova, K. Ozonat, N. Mi, J. Symons, and E. Smirni. Automated Anomaly Detection and Performance Modeling of Enterprise Applications. *ACM Trans. Comput. Syst.*, 27, 2009.
11. Compuware.com. Performance in the clouds. White paper, 2011.
12. H.P. Corporation. H.P. Openview. `http://www.openview.hp.com`.
13. M. Gallet, N. Yigitbasi, and et al. A Model for Space-correlated Failures in Large-scale Distributed Systems. In *EuroPar*, 2010.
14. Ganglia. Monitoring tool. `http://ganglia.info`.
15. Hadoop. Open source mapreduce implementation. `http://hadoop.apache.org/`.
16. IBM. Tivoli. `http://www.ibm.com/tivoli`.
17. G. Jing, J. Guofei, C. Haifeng, and H. Jiawei. Modeling Probabilistic Measurement Correlations for Problem Determination in Large-Scale Distributed Systems. In *ICDCS*, 2009.
18. G. Jung, M. Hiltunen, K. Joshi, R. Schlichting, and C. Pu. Mistral: Dynamically Managing Power, Performance, and Adaptation Cost in Cloud Infrastructures. In *ICDCS*, 2010.
19. G. Jung, K. Joshi, M. Hiltunen, R. Schlichting, and C. Pu. A Cost-sensitive Adaptation Engine for Server Consolidation of Multitier Applications. In *ACM/IFIP/USENIX Middleware*, 2009.
20. H. Kang, H. Chen, and G. Jiang. PeerWatch: A Fault Detection and Diagnosis Tool for Virtualized Consolidation Systems. In *ICAC*, 2010.
21. H. Kang, X. Zhu, and J. Wong. DAPA: Diagnosing Application Performance Anomalies for Virtualized Infrastructures. In *USENIX Hot-ICE*, 2012.
22. K. Mahendra, G. Eisenhauer, and et al. Monalytics: Online Monitoring and Analytics for Managing Large Scale Data Centers. In *ICAC*, 2010.
23. D. Mosberger and T. Jin. httperf: Tool for Measuring Web Server Performance. *SIGMETRICS Perform. Eval. Rev.*, 26.
24. Nagios. `http://www.nagios.org`.
25. K. Nesbit, N. Aggarwal, J. Laudon, and J. Smith. Fair Queuing Memory Systems. In *MICRO*, 2006.
26. H. Nguyen, Y. Tan, and X. Gu. PAL: Propagation-aware Anomaly Localization for cloud hosted distributed applications. In *USENIX SLAML*, 2011.
27. Amazon outage. Cloud disaster and recovery. `http://goo.gl/3lS13`.
28. D. Pelleg, M. Ben-Yehuda, R. Harper, L. Spainhower, and T. Adeshiyan. Vigilant: out-of-band detection of failures in virtual machines. *SIGOPS Oper. Syst. Rev.*, 2008.
29. RUBiS. E-commerce application. `http://rubis.ow2.org`.
30. Will S. and et al. Cloudstone: Multi-platform, Multi-language Benchmark and Measurement Tools for Web 2.0. In *Cloud Computing and its Application (CCA)*, 2008.
31. P. Soila and P. Narasimhan. Causes of Failure in Web Applications. Technical Report CMU-PDL-05-109, CMU, 2005.
32. SPEC. Spec cpu 2006 benchmark. `http://www.spec.org/cpu2006/`.
33. J. Tan, X. Pan, and et al. Kahuna: Problem Diagnosis for Mapreduce-based Cloud Computing Environments. In *IEEE/IFIP NOMS*, 2010.
34. Y. Tan, H. Nguyen, X. Gu, C. Venkatramani, and D. Rajan. PREPARE: Predictive Performance Anomaly Prevention for Virtualized Cloud Systems. In *ICDCS*, 2012.
35. A. Verma, P. Ahuja, and A. Neogi. pMapper: Power and Migration Cost Aware Application Placement in Virtualized Systems. In *ACM/IFIP/USENIX Middleware*, 2008.
36. A. Verma, P. Ahuja, and A. Neogi. Power-aware Dynamic Placement of HPC Applications. In *ACM ICS*, 2008.
37. A. Verma, P. De, V. Mann, T. Nayak, A. Purohit, G. Dasgupta, and R. Kothari. Brownmap: Enforcing power budget in shared data centers. In *Proc. Middleware*, 2010.
38. A. Verma, G. Kumar, R. Koller, and A. Sen. CosMig: Modeling the Impact of Reconfiguration in a Cloud. In *IEEE MASCOTS*, 2011.

39. VMware vmkperf. Monitoring tool. `http://labs.vmware.com/download/143/`.
40. VMware. Powercli cmdlets. `http://tinyurl.com/ylhrmpr`.
41. C. Wang, K. Viswanathan, L. Chodur, V. Talwar, W. Satterfield, and K. Schwan. Evaluation of Statistical Techniques for Online Anomaly Detection in Data Centers. In *IEEE IM*, 2011.
42. Chengwei Wang, V. Talwar, K. Schwan, and P. Ranganathan. Online Detection of Utility Cloud Anomalies using Metric Distributions. In *IEEE/IFIP NOMS*, 2010.
43. T. Wood, E. Cecchet, and et al. Disaster Recovery as a Cloud Service: Economic Benefits and Deployment Challenges. In *USENIX HotCloud*, 2010.