2011 30th IEEE International Symposium on Reliable Distributed Systems

CloudInsight: Shedding Light on the Cloud

Ahsan Arefin
University of Illinois at Urbana-Champaign
Urbana, IL, USA
marefin2@illinois.edu

Guofei Jiang
NEC Laboratories America, Inc.
Princeton, NJ, USA
gfj@nec-labs.com

***Abstract*—Cloud computing provides a revolutionary new computing paradigm for deploying enterprise applications and Internet services. Rather than operating their own data centers, today cloud users run their applications on the remote cloud infrastructures that are owned and managed by cloud providers. However, the cloud computing paradigm also introduces some new challenges in system management. Cloud users create virtual machine instances to run their specific application logic without knowing the underlying physical infrastructure. On the other side, cloud providers manage and operate their cloud infrastructures without knowing their customers' applications. Due to the decoupled ownership of applications and infrastructures, if a problem occurs, there is no visibility for either cloud users or providers to understand the whole context of the incident and solve it quickly. To this end, we propose a software solution, *CloudInsight*, to provide some visibility through the middle virtualization layer for both cloud users and providers to address their problems quickly. *CloudInsight* automatically tracks each VM instance's configuration status and maintains their life-cycle configuration records in a configuration management database (CMDB). When a user reports a problem, our algorithms automatically analyze CMDB to probabilistically determine the root cause and invoke a recovery process by interacting with the cloud user. Experimental results over data from Amazon EC2 online support forum and NEC Labs' research cloud infrastructures demonstrate that our approach can effectively automate the problem troubleshooting process in cloud environments.**

***Keywords*-VM; cloud computing; troubleshooting; configuration management; data analysis**

I. Introduction

The emergence and growing popularity of cloud computing signals a revolution on how IT infrastructures and services are delivered and consumed. There are many evolving cloud computing models, including Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS), which provide IT infrastructures, platforms and solution stacks, and software applications from the cloud respectively [1]. Cloud infrastructures often employ a virtualization layer to ensure resource isolation and abstraction. They are designed to provide restricted visibility to both users and IaaS providers. Users of *Virtual Machine (VM)* instances are blocked from looking down into the infrastructure layer since multiple tenants might share the same physical machine. Similarly IaaS providers such as Amazon's Elastic Computing Cloud (EC2) [2] are not allowed to look inside the running VM instances because of tenants' business confidentiality and privacy.

In a local data center, operators usually need the context information from both applications and infrastructures for effective problem determination. In commercial cloud environments, the decoupled ownership of applications and infrastructures introduces new challenges in system management. When users encounter problems in their VMs, they have little visibility on the infrastructure layer for root cause determination except *trial-and-error* troubleshooting. As discussed later, it is not even clear who is responsible for fixing such problems. To support end users, IaaS providers often run an online support forum (e.g., Amazon EC2 online forum [3]) where users can report their problems and seek solutions. Cloud operators often try to solve those problems by manually investigating the system settings of users' VMs. Since operators do not know tenants' specific applications, it is even hard for them to solve problems only based on users' problem reports. Such an approach may eventually take several hours or days to resolve problems [1].

Most of VM problems in a cloud infrastructure (except those problems caused by applications running inside VM instances) can be correlated to their configuration events. Examples of such configuration events include incorrect update of *Access Control List (ACL)* in hypervisors, blocking of access ports inside running VM instances, connection of virtual devices to unsupported OSes, migration of instances to a slow machine, incorrect allocation of memory size, connection loss with underlying block storage and so on. For example, on average 28 problems were reported daily to Amazon EC2 online forum in July 2010, complaining about VM unavailability, VM hang, Elastic Block Storage (EBS) [2] crash or loose connectivity, performance problem and so on. Problems can occur at the time when an instance is first created from its source image or during running time if its configurations are changed by some events. For example, if a running VM instance is migrated to a slow *physical machine (PM)*, a performance problem could emerge. In this case, the "PM-address" configuration of the running VM is changed by the "migration" event which causes the performance problem. Even with restricted visibility, it is possible for the infrastructure layer to track such configuration changes of VMs and correlate them with

1060-9857/11 $26.00 © 2011 IEEE
DOI 10.1109/SRDS.2011.34

problem reports.

We make two intuitive observations about cloud environments. First, multiple VM instances created from the same image source are likely to have similar characteristics in configurations. Second, some configuration attributes have higher cardinality than others and hence they are less sensitive to configuration changes. Based on those we present a solution (called *CloudInsight*) for cloud providers to automate reasoning and troubleshooting on user reported problems under the restricted visibility. *CloudInsight* runs in the infrastructure layer and tracks each VM instance as a standard object running in that infrastructure. In summary, we have made the following contributions:

Representation: *CloudInsight* develops a monitoring mechanism at the infrastructure layer to automatically capture the event history of all running VM instances in cloud environments. Each event is a change or update of the configurations of VMs as well as their underlying infrastructures. Such event data is further structured and stored in a *Configuration Management Database (CMDB)*.

Problem Reasoning: When users report a problem, our solution fetches the related information from the CMDB and further identifies a list of suspect events. We then apply statistical analysis to rank these events based on their sensitivity to configuration changes.

Interactive Troubleshooting: Following the list of ranked events, *CloudInsight* takes predefined actions to check and solve problems. For each specific event, operators define the predicates to check whether this event is the root cause as well as the actions to remediate the problem. As shown in our results, *CloudInsight* can solve a large portion of reported problems (nearly 80% of which are within the scope of cloud providers) from Amazon EC2 forum.

Experimental Evaluation: We have implemented *CloudInsight* and evaluated the solution with NEC Labs' research cloud infrastructures. In a two-month evaluation period, *CloudInsight* analyzed and diagnosed a list of user problems. We correlate these problems with user reported problems at Amazon EC2 online forum and demonstrate that *CloudInsight* can effectively identify the root cause in most cases and only take seconds to fix user problems, compared to hours or days taken in today's manual operation.

II. Motivation

In a typical cloud environment, the cloud provider is mainly responsible for the problems from its own infrastructure layer [1]. Operators of the cloud provider monitor their infrastructure stacks from the hardware layer such as servers and network devices and up to the hypervisor layer. While the cloud provider ensures a highly available cloud infrastructure, it typically does not guarantee the availability of individual VM instances since it is not even aware of what is running inside those VM instances. Therefore, cloud users will occasionally encounter various problems with their VM instances, which include VM disconnectivity, performance isolation among instances, resource disconnectivity and misconfiguration of instances.

Cloud providers offer several options to address cloud users' problems including a free best effort service with no *Service Level Agreement (SLA)* guarantee and a premium commercial grade support with SLA guarantees [1]. Both support models require cloud users to report their problems to operators, who manually investigate the problems to provide solutions. The premium support model guarantees a bounded problem resolution time with service charges while the best effort model is free but might take 8 to 10 days to solve user problems [1]. Hence, our objective is to provide automated solutions for cloud providers to address users' problem quickly and also reduce the cost of manual operations. Our experimental results show that *CloudInsight* can effectively reduce the solution time to seconds.

Many problems may manifest themselves in a similar phenomenon and it is not easy to determine the root cause based on user reports. When an instance is unreachable, there could be many reasons behind that, such as instance crash, incorrect ACL, hardware issues and so on. Benson *et al.* [1] categorized user reported problems of Amazon EC2 into five different classes. As reported in their analyses on 3-year operation data, application and image maintenance related problem reports are decreasing over time due to the increased user familiarity with cloud infrastructure. The percentage of connectivity problem reports (due to firewall or IP issues etc.) is nearly constant, and problem reports related to virtual infrastructure and performance are increasing over time. Therefore, *CloudInsight* aims to provide automated solutions to resolve connectivity, performance and virtual infrastructure related problems in the cloud. Our analysis on Amazon EC2 forum over the month of July 2010 shows that *CloudInsight* covers nearly 80% of the reported problems which are within the management responsibility of IaaS cloud providers. However, many problems reported in Amazon EC2 request to restart/shutdown of hanged/crashed instances, which have to be confirmed by users. *CloudInsight* addresses this by communicating with end users through an interactive troubleshooting process.

III. *CloudInsight* Models and Architecture

A. System Models

Problem Models: *CloudInsight* aims to solve connectivity, infrastructure and performance related problems by automating the problem reasoning process. We classify these problems into six classes according to their root causes.Table I shows the name and definition of these problem classes. They can be combined to characterize real problems. For example, an instance can crash (a VM Availability problem) because of errors in connected virtual devices (a VD

Misconfiguration problem). Our analysis on Amazon EC2 user forum (in Section V) concludes that these problem models cover approximately 66% of all reported problems and 80% of the problems that are within the management responsibility of cloud providers.

Table I
PROBLEM MODELS.

Problem class	Problem definition
PM Performance	Related to PM performance
PM Connectivity	Related to PM failures
VM Connectivity	Related to instance reachability
VM Availability	Related to VM instance crash
VD Misconfiguration	Related to virtual devices
VI Misconfiguration	Related to virtual interfaces

Data Models: *CloudInsight* stores all the configuration parameters of running VM instances as well as those of physical machines in a database. Table II shows the examples of the configuration attributes of VMs and PMs, which can be monitored and tracked at the hypervisor layer. Different cloud infrastructures may add their own set of specific attributes. At time t, the set of configuration attributes of a running VM α is a vector $[x_0, x_1, ..., x_{(N-1)}]$, where x_i is the i^{th} $(0 \leq i < N)$ attribute. α is the unique identification of a running VM instance. When a VM is created, it is assigned with a 128 bit unique identifier called $uuid$ and a VM maintains this $uuid$ even after its migration. *CloudInsight* uses this $uuid$ to track VM instances during their life-cycle. Similarly we define the configuration set of a PM with ID defined by its IP address.

Table II
DATA MODELS (*=MULTIPLE ENTRY FIELD).

Configuration type	Attributes
VM configuration	*uuid, mac, ip, instance-name, source-img, source-img-size, OS, port, memory, vcpu, virtual-device(vd)*, virtual-interface(vi)*, PM-address, uptime, instance-status,* etc.
PM configuration	*hostname, ip, OS, cpu*, memory, number-vms*, etc.

Event Models: Any change in the configuration set is considered as an event in this paper. For example, the creation and termination of VM instances are such events which change the number of running instances ($number\text{-}vms$ attribute in Table II) on the host PM; A VM migration is another event that not only changes the number of running instances on their host PMs but also changes the "PM-address" attribute of the migrated VM instance.

B. System Architecture

The system architecture of *CloudInsight* is shown in Figure 1 and it consists of three major components: *Monitoring Agent*, *Cloud CMDB* and *Analysis Engine*.

Monitoring Agent: The Monitoring Agent monitors and tracks the configuration attributes of VM instances and

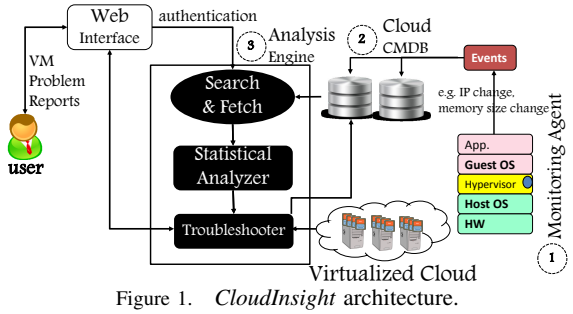

Figure 1. *CloudInsight* architecture.

infrastructures continuously. We assume that all PM clocks are loosely synchronized. The agent sits in the hypervisor layer so that it can be rolled out through the cloud infrastructures as a standard management component. Section IV-A discusses the details of data collection.

Cloud CMDB: CMDB Manager structures and stores configuration data sent by the monitoring agent into a database called Configuration Management Database (CMDB). CMDB is often used for asset and configuration management in large data centers.In this paper, we use a CMDB to track the life-cycle configuration information of VM instances. It is a relational database and each row includes a configuration change along with its timestamp and health status (*sick* or *healthy*). Initially all rows are marked as healthy. If a configuration change is found to be the cause of a reported problem, the row containing this change is marked as sick in the database during the troubleshooting step. In Section IV-B, we discuss how CMDB is designed and populated in cloud infrastructures.

Analysis Engine: After users encounter a problem with their VM instances, they report the problem to *CloudInsight* via a web interface. The Analysis Engine fetches the data related to the reported VM from the CMDB and compares its current configuration set with its historical records to determine a list of suspect events. Then the engine employs statistical analysis to rank the list of suspect events and uses predefined predicates and actions to check these events and resolve the problem. Not all problems can be automatically resolved by *CloudInsight* since some problems are beyond the management responsibility of cloud providers. However, the suspected events are reported to end users with an interactive troubleshooting process and its design details are given in Section IV-C.

IV. DESIGN AND IMPLEMENTATION

A. Data Collection

CloudInsight runs an event-based monitoring agent (ci_agent) at the hypervisor layer of each physical machine. The agent is a JAVA program that collects system configuration data using Linux shell scripts. The shell scripts use $virsh$ management interface libraryto collect values of configuration attributes of running VM instances such as $uuid$,

mac, $vcpu$, $memory$, OS, $uptime$, $virtual\ interfaces$, and $virtual\ devices$. ci_agent also collects IP addresses of running guests by looking into the system *arp-cache*. Other information about hypervisor, physical machine and open ports in the running instances are collected using Linux standard shell commands.

Collected information are represented using an XML format. Configuration data is collected periodically at an interval of τ, but it is not sent to the CMDB Manager unless there is a change in the configuration data compared to the previous one, which indicates new events. Hence the data collection is event-based to reduce network overhead. τ is defined by infrastructure operators. τ bounds the difference between the occurring time and logging time of an event. A large value of τ may result in missing events. Conversely a small value of τ would increase the number of repeated polling, which may lead to higher overhead in the hypervisor. However, our experimental results show that ci_agent introduces very little overhead on the hypervisor even with a very small τ=1sec.

B. Populate CMDB

The central CMDB Manager node runs a $ci\_manager$, a JAVA program that collects event-based configuration data from monitoring agents. It parses collected XML messages and uses a relational database to structure and store all configuration data (shown in Table II) along with the reported timestamp. Each row in the database indicates at least a single change in the set of configuration attributes. As we mentioned before, the number of attributes may vary with various cloud infrastructures. Each row in the database is also associated with a status tag that defines whether the configuration is sick or healthy. Initially all entries are set to be healthy and a configuration is changed to be sick only if it is proved to be faulty during the troubleshooting process.

An example of the configuration records of a VM instance ($\alpha = VM0$) is given in Table III, which shows 6 attributes ($uuid$, $source\text{-}img$, $PM\text{-}address$, $memory$, vd and $port$) along with their timestamp. The first row indicates an event when the VM is restarted at physical node $x.x.164.148$. The configuration changes over time are shown in bold at the subsequent rows. The second row corresponds to a migration that migrates the instance to $x.x.164.150$. The third row indicates a change in the memory size as well as the addition of a new virtual devices EBS-1235246. The last row shows the change in the list of opened ports (this information is necessary as EC2 requires specific ports to keep open). CMDB stores similar tables for PMs and their attributes are shown in Table II.

C. CMDB Analysis

When a problem is reported, the analysis engine analyzes the related CMDB records of the reported VM instance to determine the suspect events and then tries to check the root cause and solve the problem. The whole process is divided into three steps: *Identifying Suspect Events*, *Event Ranking* and finally *Interactive Troubleshooting*.

1) Identifying Suspect Events: As discussed earlier, when a problem of a VM instance occurs, it is most likely caused by the recent events of that VM instance. As the CMDB maintains the complete event history of VM instances, the analysis engine can determine the list of suspect events from the CMDB. Each event is associated with a change in the configuration attribute called *suspect attribute*. *CloudInsight* identifies the changes of the reported instance by comparing its current configuration to the latest known *stable* configuration. Any configuration that persists longer than a specific duration of time (ΔT) is considered as a stable configuration. Unstable configurations may arise when users try to fix problems by trial and error so that every configuration change only lasts for a very short period of time ($< \Delta T$).

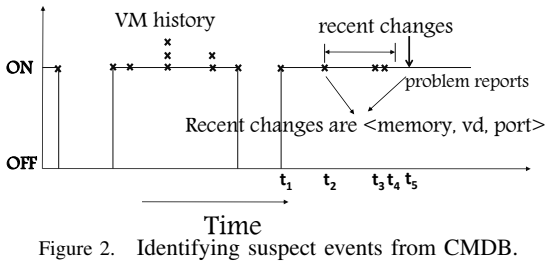

Figure 2. Identifying suspect events from CMDB.

Let's consider the previous example in Table III. We draw the event history of VM0 as shown in Figure 2. Each cross represents a single event and 13 different events are shown in the figure when VM0 is in ON state. Assume that a problem is reported at time t_5. The current configuration is the one updated at time t_4. We assume $t_4 - t_3 < \Delta T$ and $t_3 - t_2 \geq \Delta T$. Hence configuration updated at time t_3 is unstable and we consider t_2 as the time point with the latest stable configuration. *CloudInsight* then identifies the set of suspect events by comparing the configuration changes between time t_5 and t_2. In this example the suspect events are the changes of attributes $port$, vd and $memory$. If we use χ to define the list of suspect events, then $\chi = [port, vd, memory]$. For cloud infrastructures, this set can be quite large.

2) Event Ranking: After determining the set of suspect events, *CloudInsight* ranks them based on how likely an event could be the problem root cause. As the set of suspect events can be large, we reduce the problem resolution time by checking the events in their ranked order. An event is highly suspected if it is very rare to occur. To measure this, we define a sensitivity metric and rank the set of suspect events according to their sensitivity, i.e., the sensitivity of an attribute to change.

We define two types of event sensitivity: *local sensitivity* and *global sensitivity*. The local sensitivity defines the probability of an event's occurrence using the event

Table III
AN EXAMPLE OF VM CONFIGURATION RECORDS.

timestamp	*uuid*	*source-img*	*PM-address*	*memory*	*vd*	*port*
2010-08-09 04:39:50 (t_1)	1813b7d4-c49f-831d-6b5c-98988733e824	/var/lib/xen/images/as217.img	x.x.164.148	524288	CD-2051473282	22,80
2010-08-09 05:29:10 (t_2)	1813b7d4-c49f-831d-6b5c-98988733e824	/var/lib/xen/images/as217.img	**x.x.164.150**	524288	CD-2051473282	22,80
2010-08-09 05:43:23 (t_3)	1813b7d4-c49f-831d-6b5c-98988733e824	/var/lib/xen/images/as217.img	x.x.164.150	**524000**	**CD-2051473282, EBS-1235246**	22, 80
2010-08-09 05:45:23 (t_4)	1813b7d4-c49f-831d-6b5c-98988733e824	/var/lib/xen/images/as217.img	x.x.164.150	524000	CD-2051473282, EBS-1235246	**22**

history of the instances originated from the same image source. All instances originated from the same image source should have very similar characteristics because they have the same application logic, guest OS and so on. If $Src(\alpha)$ represents the image source of the instance α, the local sensitivity of an attribute is calculated using the records of all instances originating from $Src(\alpha)$. Meantime the global sensitivity defines the probability of an event's occurrence using the event history of all VM instances regardless of their image sources. The characteristics of some attributes are not dependent on specific image types and hence we can use the data from the larger global VM population to calculate the sensitivity of an attribute.

To calculate two types of sensitivity, we extract two data sets from the CMDB: 1) *local configuration data set* ($cmdb_l$) that includes the event history of all instances originated from the same source $Src(\alpha)$ of the problematic instance α, 2) *global configuration data set* ($cmdb_g$) that includes the event history of all instances. Note that for both data sets, we only extract the records of suspect attributes in χ.

The sensitivity of events should be considered locally and/or globally. An event may not be allowed for a specific image while it is very common for other images. Therefore it is misleading to rank this event using only global sensitivity. Conversely, an event is very common for a specific image while it is not allowed for many other images. To this end, we introduce a weighted variable θ to define how closely the statistics of an attribute from $cmdb_l$ matches that from $cmdb_g$. The value of θ ranges from 0 to 1. Given an attribute, if its statistics from $cmdb_l$ is very similar to that from $cmdb_g$, we have $\theta = 1$ and use the global sensitivity for its ranking. On the other side, if its statistics from $cmdb_l$ is very different with that from $cmdb_g$, we have $\theta = 0$ and use local sensitivity for its ranking. Both cases are the extreme cases. Later in this section we discuss how to calculate θ for common cases which need the weighted combination of global and local sensitivity for its ranking.

The sensitivity of an attribute can be defined by $P(S|M)$, the probability that an attribute's modification (M) causes the problem (S). We estimate this probability for all suspect attributes in χ based on the local data set ($cmdb_l$) and global data set ($cmdb_g$). For simplicity, let us derive the probability $P(S|M)$ for the i^{th} suspect attribute and all the following parameters should be implicitly indexed by i. The probability $P(S|M)$ can be defined with the following equation:

$$P(S|M) = (1-\theta)P(S|M)_{cmdb_l} + \theta P(S|M)_{cmdb_g} \quad (1)$$

The conditional probabilities for local data set (defined by $P(S|M)_{cmdb_l}$) and global dataset (defined by $P(S|M)_{cmdb_g}$) can be derived with the following equation by Bayes rule [4] (where $X = P(M|S)_{cmdb}P(S)_{cmdb}$):

$$P(S|M)_{cmdb} = \frac{X}{X + P(M|\bar{S})_{cmdb}P(\bar{S})_{cmdb}} \quad (2)$$

$\bar{S}$ represents that the suspect attribute does not cause the reported problem. We need to estimate each of the terms on the right side of Equation (2). We assume that a problem is only caused by one attribute change at a time. We skip the subscript here since we need to calculate the probabilities for both local and global data sets. The prior probability on whether an attribute in the suspect set χ causes the problem or not can be defined as $P(S) = \frac{1}{n}$ and $P(\bar{S}) = 1 - \frac{1}{n}$, where n is the number of suspect attributes in χ.

In practice, since most VM instances are healthy, we do not have much historical data about the sick configurations of a specific VM instance, especially when it encounters a problem for the first time. Therefore we assume all attributes have the equal probability to change when a problem occurs and hence $P(M|S) = \frac{1}{|C|}$, where $|C|$ is the cardinality of the configuration attribute set in CMDB.

Given a set of suspect events, we have the same $P(S) = \frac{1}{n}$, $P(\bar{S}) = 1 - \frac{1}{n}$ and $P(M|S) = \frac{1}{|C|}$ for all of these attributes while the value of $P(M|\bar{S})$ varies for different attributes. Therefore Equation (2) can be reformulated as

$$P(S|M_{cmdb}) = \frac{1}{1 + |C|(n-1)P(M|\bar{S})_{cmdb}} \quad (3)$$

In the following subsections, we discuss how to calculate $P(M|\bar{S})$ with local and global data sets respectively.

Calculating $P(S|M)_{cmdb_l}$: To calculate $P(S|M)_{cmdb_l}$, we define a local vector L of size n (the number of suspect events). Each element $L[\nu]$ represents the number of unique values that the ν^{th} suspect attribute (in χ) takes in all instances originated from $Src(\alpha)$. In other words, each element in L represents the number of unique values that a suspect attribute takes in the local data set. Continuing the previous example shown in Table III, let's assume that we have $L = [8, 7, 5]$, where $n = 3$ and $\chi = [memory, vd, port]$. The element $L[0] = 8$ refers that all instances originated from the source image $Src(VM0)$ take

8 different values of $memory$ configuration during their life cycle. Meantime $L[3] = 5$ means that these instances take 5 different values in $port$ configuration. Note that these instances may not be alive at current time and the vector L is also calculated only from the healthy configuration records of the CMDB.

For the suspect attribute z_i (the i^{th} attribute in χ), the local cardinality of z_i is $|z_i|_l = L[i]$. The total number of different values that all suspect attributes in χ take in the local data set is $L_\chi = \sum_{\nu=1}^{n} L[\nu]$. Thus, $P(M|\bar{S})$ can be computed with the equation:

$$P(M|\bar{S}) = \frac{|z_i|_l}{L_\chi} \tag{4}$$

Calculating $P(S|M)_{cmdb_g}$: Similarly to calculate the probability $P(S|M)_{cmdb_g}$, we define a global vector G of size n. G is also calculated only with the healthy configuration records from the global data set. Each element $G[\nu]$ in vector G represents the number of unique values that the ν^{th} suspect attribute (in χ) takes in all instances of the global data set. In the previous example, the G vector can be represented as $G = [32, 37, 18]$, where $n = 3$ and $\chi = [memory, vd, port]$. The element $G[0] = 32$ refers that all instances (even from different source images) in the global data set take 32 different values of $memory$ configuration during their life cycle.

In a similar way, the global cardinality of z_i is $|z_i|_g = G[i]$ and the total number of unique values that all suspect attributes take in the global data set is $G_\chi = \sum_{\nu=1}^{n} G[\nu]$. Thus we can define equation:

$$P(M|\bar{S}) = \frac{|z_i|_g}{G_\chi} \tag{5}$$

Calculating θ: To calculate θ, we define a (p X q) dimensional matrix D, where p is the number of images in the global data set and q is the number of suspect attributes. Assume that the set of all source images in the CMDB is I. Thus each element $D[\mu][\nu]$ in the matrix D represents the cardinality of the ν^{th} suspect attribute in all instances from the μ^{th} image, $0 \leq \nu < q$ and $0 \leq \mu < p$. Table IV shows the corresponding D matrix in the previous example, where I=[$as213.img$, $as214.img$, $as215.img$, $as216.img$], $p = 4$ and $q = n = 3$. Thus $D[1][0] = 10$ means that all instances originated from the image $as214.img$ take 10 different $memory$ configurations. The CMDB may only hold the most recent records in a month or a quarter (rather than a much longer time period) to reduce the database size. Meantime D can be updated only once per day since the distribution of all historical records does not change much everyday.

As we discussed earlier, we define θ to measure how closely $cmdb_l$ matches $cmdb_g$ for a specific event. In other words, it is defined as the probability on how the statistics of an event in the local data set ($cmdb_l$) follows that in the global data set ($cmdb_g$). To compute θ for a suspect attribute,

Table IV
AN EXAMPLE OF THE MATRIX D.

image-id	*memory*	*vd*	*port*
as213.img	8	7	5
as214.img	10	12	2
as215.img	10	15	3
as216.img	4	5	8

we build a distribution from D by taking the column values of that suspect attribute across all VM images. Then θ is the probability on how the L value (from a specific VM image) of a suspect attribute falls in its distribution constructed by the corresponding column values (across all VM images) in D matrix.

Table V
AN EXAMPLE OF PROBABILITY CALCULATION.

Attribute	$P(S\|M)_{cmdb_l}$	$P(S\|M)_{cmdb_g}$	θ	$P(S\|M)$
$memory$	0.073	0.076	0.75	0.075
vd	0.082	0.067	0.75	0.071
$port$	0.11	0.128	0.5	0.119

With Equation (4), (5) and θ, we can further follow Equation (3) and (1) to calculate the probability $P(S|M)$ and rank all suspect attributes. Let's consider the previous example: We have χ=[$memory$, vd, $port$] for the instance VM0. The value of L, G and D are shown in the above tables respectively. The conditional probabilities and the θ values are shown in Table V. Following Equation (3) and Equation (1), we can calculate the exact value of $P(S|M)$ respectively. If we compare the $P(S|M)$ values of these suspect attributes, we can rank them with the following sensitivity order (from high to low): χ_r=[$port$, $memory$, vd].

3) Troubleshooting: After determining and ranking suspect events, *CloudInsight* performs troubleshooting actions, which include two major steps:**Check** and **Solve**. For each suspect event, the check step checks whether it is the true root cause (or culprit event) of the problem. The solve step tries to solve the problem automatically if the solution is within the management responsibility of the cloud provider, otherwise *CloudInsight* recommends the solutions to the cloud users.

To identify the root cause, *CloudInsight* uses a series of predefined predicates to check each suspect event, starting from the top of the ranked suspect events. Predicate based probings are getting popular for troubleshooting in the recent literature [5]. The suspect events are first classified into one or more problem classes. Actions are defined for each individual problem class and also include a series of predicates to check the existence of the corresponding problem. In fact it is very convenient to run check and solve steps in cloud infrastructure due to the availability of temporary "tested". For example, cloud infrastructure has elastic virtualized resources, which enables us to start, migrate and remove VM instances quickly. This allows us to build a temporary testbed for problem verification using the existing virtualized cloud resources. The check step also

uses PM configurations to validate the current configuration of reported VM instances. If the check fails, then the event is not the root cause and it is removed from the list of suspect events, otherwise *CloudInsight* moves into the corresponding solve step.

The solve step uses predefined solutions for the problem validated at the check step. Sometimes the solution is not within the scope and management responsibility of the cloud provider. In that case, the solution is reported to the users so that they can apply them inside their VM instances. As the check step removes the *non-culprit* events, it reduces the set of suspect events so that eventually users get a small list of events containing the true root cause.

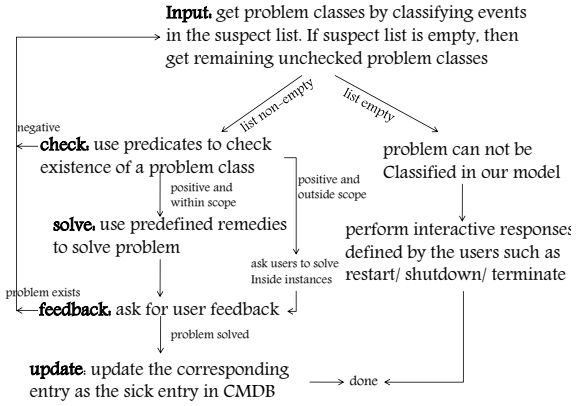

Figure 3. *Check-and-solve* flow chart.

The whole check-and-solve troubleshooting process is done interactively with users. After each solve step, users are asked to provide feedbacks. If the problem is solved, the troubleshooting process ends; otherwise the check-and-solve steps continue for the next suspected event. This process is very similar to the troubleshooting support for desktop printing or networking in Windows OS. Sometimes the check-and-solve steps need to shutdown or restart a instances to solve the problem, which has to get user's confirmation. If some users want to run their instances without any interruption, *CloudInsight* sends the solution back to the users for their own decision. After a problem is reported, *CloudInsight* automatically invokes problem reasoning and uses the same web interface to interact with users and acquire their confirmation and permission for actions. In some cases, our check and solve approach might not be able to solve user reported problems. After several rounds of check and solve steps, the list of suspect events becomes empty and the problem is still not solved. For such cases, based on users' preference, *CloudInsight* can notify operators to follow up a manual investigation or continue to check other problem classes. Figure 3 shows the complete flow chart of our check-and-solve troubleshooting algorithm.

Here we use the previous example to illustrate the troubleshooting process. The output of suspect event ranking is χ_r=[$port$, $memory$, vd]. At first, *CloudInsight* classifies these suspect events into problem classes. Changing $port$ event is classified into VM Connectivity, changing $memory$ event is classified into PM Performance and VM Availability, and changing vd event is classified into VD Misconfiguration and VM Availability. As an example, Table VI shows the series of predicates for the check-and-solve steps of PM Performance. Similarly, we define check-and-solve steps for other problem classes.

Table VI
Check-and-solve ACTIONS FOR PM PERFORMANCE.

Actions
• Get the current PM of the reported VM using $PM\text{-}address$ • Get $cpu\text{-}usage$ and $memory\text{-}usage$ using SNMP • IF $cpu\text{-}usage > threshold$ or $memory\text{-}usage > threshold$ THEN there is a performance problem • Migrate VM to different PM • IF migration fails THEN re-create/shutdown the VM in different PM with current configuration (depending on users choice)

As we see in the Table VI, for measuring the performance of physical machines, *CloudInsight* requires each physical machine to run a lightweight SNMPclient in the hypervisor. During the check step, an SNMP request is issued to collect CPU and memory information of a physical machine to check for its performance. The performance is considered good if CPU and memory usages are below a defined threshold. If a performance problem is detected, the solve step migrates the user instance out of the busy machine to a different physical machine. Note that we only use Table VI as a simple example to illustrate the check and solve process. In practice, operators can define much more sophisticated predicates and actions to check and solve common problems.

V. SYSTEM EVALUATION

A. Experimental Setup

We have implemented the *CloudInsight* troubleshooting system. The CMDB manager uses a MySQL database and runs on a server with Intel Xeon 2.4GHz CPU and 2GB RAM. We use 29 physical machines of NEC Labs' research cloud infrastructures to collect data. Each machine is virtualized with Fedora Xen. Five research projects are using these machines for cloud and virtualization related research experiments. Researchers have frequently created, changed, and migrated their VM instances for their experiments. We collected data from 1^{st} July 2010 to 15^{th} Aug 2010. The number of VMs varies since researchers create and stop VM instances frequently for their experiments. In total, we collected 358 events for 138 distinct VM instances originated from 32 distinct image sources. We use $\tau = 1$sec during our experiments unless specified. Communication between ci_agent and $ci\_manager$ is done using TCP to avoid message loss and the synchronization of PM clocks is done using NTP [6].

Table VII
RELATED PROBLEM REPORTS AT AMAZON EC2.

Problem reports in our infrastructure	*Similar reports in Amazon EC2*
Low Performance	• "Instance not responding to SSH" [07/24/10] • "Instance cant access instance metadata by HTTP protocol" [07/13/10] • "EBS bad performance" [01/01/10]
Instance not accessible (PM crash)	• "EC2 instance wont respond over SSH, system log not updating" [07/22/10] • "Instance not responding" [12/24/09]
Instance not accessible (VM crash)	• "Instance and EBS crash" [07/17/10] • "Cannot stop, force stop or reboot i-635d0108" [07/27/10]
Instance not accessible (storage volume hang)	• "Cannot detach a volume" [02/11/10] • "Cant connect to instance, connection timeout" [07/21/10]
Instance not accessible (misconfigured firewall)	• "I cant SSH to my instance i-obo7d960" [07/10/10] • "I cant connect to my EC2 instance, whats the matter?" [07/27/10]
Instance not accessible (misconfigured DNS)	• "Need support to change internal IP" [07/27/10] • "Cannot remote connect after change to remote IP" [05/04/09]

Table VIII
ROOT-CAUSES OF USER REPORTED PROBLEMS (FOR OUR INFRASTRUCTURE AND CORRELATED EC2 FORUM MESSAGE).

Problem report	*Root-cause*	*Culprit event*
Low Performance	• Instance migration to a slow machine (1) • Creation/migration of other instances to the current machine to overload • Increasing memory size of current instance	• change of $PM\text{-}address$ • change of $PM\text{-}address$ • change of $memory$
PM crash	• Hardware failure (not related to any configurations) (2)	• H/W failure
VM crash	• VM crash by changing OS kernel (3) • Application related (not related to any configurations)	• change of os • crash by applications inside
Storage volume hang	• Attaching/detaching EBS (we use virtual CD-rom) (4)	• change of vd
Misconfigured firewall	• Application blocked ports ($5a$) • Migration blocked ports due to non-updated ACL ($5b$)	• change of $port$ • change of $PM\text{-}address$
Misconfigured DNS	• Change IP (6) • Change MAC	• change of ip • change of mac

B. Problem Characteristics

During our evaluation period, researchers reported their VM related problems from time to time. When a problem was reported, we first analyzed our cloud environment manually to identify the root-cause. This information is used as the "ground truth" to verify the correctness of *CloudInsight* problem reasoning. To correlate these problems with those from commercial clouds, we analyzed problem threads from Amazon EC2 online support forum using a JAVA-based crawler. The crawler collects the posted threads by following the "subject line" of messages. The root-causes of these threads are manually identified by analyzing the follow-up conversations in the message threads. Table VII summarizes the list of problems from our research infrastructure as well as their correlated problems from Amazon EC2. The second column shows the "subject line" of the related EC2 problem threads along with their report date. The root causes and corresponding culprit events of these problems are shown in Table VIII. We assign a ID for each problem (inside "()" of the second column in Table VIII) and use the problem IDs in the following discussion.

C. CloudInsight *Performance*

1) Root Cause Ranking: After a problem is reported by researchers, *CloudInsight* determines a list of suspect events and each suspect event changes a single suspect attribute. For each problem, the list of suspect attributes determined by *CloudInsight* is shown in Table IX. Note that the problem IDs are labeled in the second column of Table VIII. The third column shows the culprit attribute that originated the problem. After determining the list of suspect events, *CloudInsight* pulls the related data from the CMDB and calculates a local vector (L) and a global vector (G). A global matrix D is also created to compute θ. The size of D in our experiments is shown in the forth column of Table IX.

Table IX
IDENTIFYING AND RANKING OF SUSPECT EVENTS.

Problem	*Suspect attributes*	*Culprit attribute*	D
1	$memory$, $vcpu$, vd, $PM\text{-}address$	$PM\text{-}address$	(32X4)
2	$memory$, $vcpu$, $device$, OS, ip	None (H/W fail)	(32X5)
3	OS, $memory$, $vcpu$, vd, ip	OS	(32X5)
4	vd, $memory$	vd	(32X2)
$5a$	$port$, $memory$, $device$, OS, ip	$port$	(32X5)
$5b$	$PM\text{-}address$	$PM\text{-}address$	(32X1)
6	ip, $vcpu$, $port$	ip	(32X3)

Finally Equation (1) is used to calculate and rank the sensitivity of the suspect events. Table X shows the rank of

the culprit event in each problem. In most of the 7 problems, *CloudInsight* accurately identified the culprit event as the first in ranking. Only in the case of problem no. 2, *CloudInsight* can not find any suspect events since the H/W failure is not detected.

2) Troubleshooting: An interactive troubleshooting is invoked for each problem. We use 2 extra machines from the research cloud infrastructure to support problem troubleshooting. Therefore we can migrate/clone the problematic instances to these extra machines for check and solve. In our experiments, most problems are solved in seconds while Amazon EC2 requires hours or days to resolve similar problems, as shown in Table X.

Table X
TROUBLESHOOTING WITH *CloudInsight*.

Problem	*Rank of culprit attribute*	*Amazon sol. time*	*CloudInsight sol. time*
1	2	7 days	9.965 sec
2	X	13 hrs	12.873 sec
3	1	16 hrs	12.223 sec
4	1	1 day	19.245 sec
$5a$	1	3 day	0.423 sec
$5b$	1	1 day	0.582 sec
6	3	15 hrs	0.865 sec

In most of the cases (except for case 2), the suspect events are found to be the cause of the problems. In case 1, 4 and $5b$, the reported problems are solved automatically by *CloudInsight*. For the problems in case 3, $5a$ and 6, *CloudInsight* provides the solutions to the users for their own decision. *CloudInsight* recommends the users to change guest OS kernel to avoid incompatibility issues in case 3, open ports in case $5a$ and fix IP address in case 6. For the case of H/W failure, as the problem persists even after all suspect events are checked, *CloudInsight* further checks the predicates of the remaining problem classes. The predicates in PM Availability problem class eventually find the problem and *CloudInsight* resolves it by re-creating the instance at a different PM. Such check-and-solve based troubleshooting steps are quite simple and do not lead to much computation and communication overhead.

3) CloudInsight *Overhead:* As a management tool, *CloudInsight* only has very light computation and communication overhead. In *CloudInsight*, each physical machine runs only one monitoring agent ci_agent at the hypervisor. As data is only sent out for new events, its overhead mainly comes from the polling of monitored attributes periodically. As we can see in Figure 4(a), the CPU overhead is very low even with 1sec of polling interval (τ) and it is nearly zero for τ=10sec. The overhead increases with the number of VM instances on the host machine due to the increase of querying time to capture configuration data. However, the number of VMs hosted by a single host machine is very limited by its hardware resource so that the monitoring overhead of *CloudInsight* remains very small.

Data is sent to the central CMDB manager only if ci_agent detects a new event. The XML message only contains the configuration changes of that event along with the identities of related PMs and VMs, which results in very low communication overhead. We measure the average number of bytes per message during our data collection period. As monitoring agents poll at the interval of τ, the number of events can vary during this interval. The average communication overhead for different number of events is shown in Figure 4(b). For a commercial cloud, this amount of network traffic is fairly low. The overhead also increases with the larger number of events enclosed in the message. For $10,000$ physical machines, even with 6 events (i.e. configuration changes) per machine during τ=10sec, the total network traffic will be around 429KBps. Note that the total traffic overhead of *CloudInsight* is simply linear with the number of physical machines and the number of events per machine per sampling time.

D. Problem coverage of CloudInsight

CloudInsight provides automated solutions for all problems that belong to our six problem classes. One question is how many real user problems belong to these six classes (hence the coverage of *CloudInsight*).

To address this, we collected and analyzed all online message threads posted in the month of July 2010 from Amazon EC2 support forum. 816 separate message threads are collected and clustered into 32 categories. We use an information retrieval tool called Carrot2 [7] with Lingo [8] clustering algorithm to cluster the message threads. To interpret these 32 clusters, we generate a *summary* of each cluster, which consists of the top 20 words (in terms of frequency counts) from each cluster. Using this summary, we define five disjoint groups out of these 32 clusters. All clusters with application related problems are grouped into "Application" group, which is about 13% of total problem reports. Similarly all clusters related to our six problem classes are grouped into "CloudInsight" group. It is nearly 66% of the total problem reports. About 2% online messages posted questions or comments about the management software such as CloudWatch, which are grouped into "Management" group. 33 messages are related to non-technical questions including billing queries, future releases and feature requests. We group them into "Question and Help" category. *CloudInsight* still does not provide solutions for network QoS, password or load balancing related issues. About 120 problems were reported in this category and they are grouped into "Net/ELB/Image/PW" group. Our classification is shown in Figure 4(c). Note that only "CloudInsight", "Net/ELB/Image/PW", and "Management" groups are within the scope and responsibility of cloud providers and *CloudInsight* can solve nearly 80% of these problems automatically.

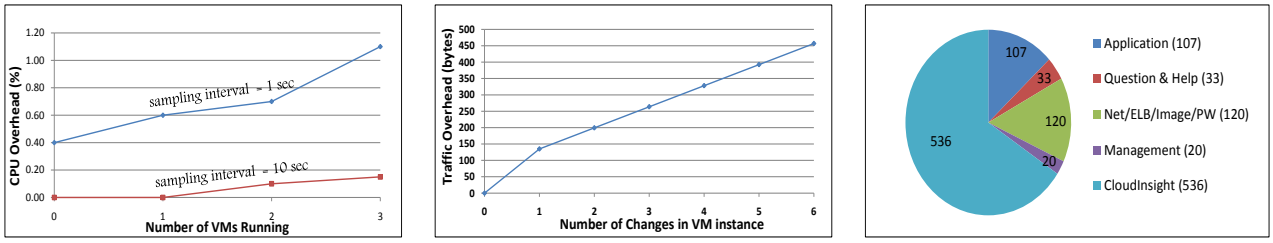

Figure 4. (a) CPU overhead of *CloudInsight*, (b) Traffic overhead of *CloudInsight*, and (c) Problem coverage of *CloudInsight*.

VI. Related Work

Much work has applied statistical learning methods to detect and diagnose problems in large-scale computer systems. Cohen *et al.* [9], [10] used a tree-augmented naive (TAN) bayesian network to learn the probabilistic relationship between SLA violations and system resource usages. They used this learned bayesian network to identify performance bottlenecks. Bodik *et al.* [11] proposed a "Fingerprint" approach, which identifies the problem with performance fingerprinting. The authors proposed a method to correlate key performance indicators (KPIs) with system SLA. Several other works [12], [13] took similar statistical analysis approach to diagnose problems from system monitoring data. Jiang *et al.* [14] developed an invariant-based approach to profile large systems for management purpose. These approaches seem to work well for performance related problems in local data centers. However, such performance problems only contribute to 11% of user reported problems in Amazon EC2 [1]. Also due to the highly dynamic nature of cloud environments, learning based methods are less effective. Unlike others, *CloudInsight* monitors a set of VM configuration attributes, and addresses a wider class of problems realted to virtual instances such as virtual device incompatibility, blocking ports or OS incompatibility etc..

There also exist some literature on configuration management. Whitaker *et al.* [15] developed a tool named Chrouns to automatically search for a configuration state change indicating a failure. It stores local system configurations at each checkpoint along time. When a problem occurs, Chronus takes user-provided probes to identify the working and non-working configuration state. This approach is quite similar to our check and solve step. However, *CloudInsight* works for troubleshooting VM instances in cloud infrastructure while Choruns is only designed for troubleshooting a single machine. Su *et al.* [5] also proposed an approach called "autobash" for troubleshooting applications on a single machine. Wang *et al.* [16] proposed a method called "PeerPressure" to support automatic troubleshooting of misconfiguration in Windows registry file. Their problem domain is different from ours because they use healthy windows registry files to determine the misconfiguration though both of our approaches use Bayesian rules. *CloudInsight* collects the attribute records from running VM instances and cloud infrastructure, and the problem reasoning is based on these event records.

VII. Conclusion

In this paper, we present *CloudInsight*, a novel solution for automated problem troubleshooting in cloud environments. *CloudInsight* monitors and tracks the configuration attributes of VM instances and infrastructure, and uses the historical event records to determine the root cause of problematic VM instances, and further provides a check-and-solve troubleshooting process to resolve user reported problems automatically. In our future work, we plan to monitor more configuration attributes from cloud infrastructure and design new predicates and actions to cover more problem classes.

References

[1] T. Benson, S. Sahu, A. Akella, and A. Shaikh, "A first look at problems in the cloud," in *Proc. of HotCloud*, 2010.
[2] Amazon EC2, "http://aws.amazon.com/ec2/."
[3] Amazon EC2 Forum, "http://developer. amazonwebservices. com/connect/forum.jspa?forumid=30."
[4] A. Gelman, J. Carlin, H. Stern, and D. Rubin, "Bayesian data analysis," *Chapman*, 1995.
[5] Y. Su, M. Attariyan, and J. Flinn, "Autobash: Improving configuration management with operating system causality analysis," in *Proc. of SOSP*, 2007.
[6] D. Mills, "Network time protocol (version 3) specification, implementation," in *IETF Draft Standard RFC-1305*, 1992.
[7] Carrot2, http://project.carrot2.org/.
[8] S. Osinski, J. Stefanowski, and D. Weiss, "Lingo: Search results clustering algorithm based on singular value decomposition," in *Proc. of IIPWM*, 2004.
[9] I. Cohen, M. Goldszmidt, T. Kelly, and J. Symons, "Correlating instrumentation data to system states: A building block for automated diagnosis and control," in *Proc. of OSDI*, 2004.
[10] M. Goldszmidt, A. Fox, I. Cohen, J. Symons, S. Zhang, and T. Kelly, "Capturing, indexing, clustering, and retrieving system history," in *Proc. of SOSP'05*, 2005.
[11] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: automated classification of performance crises," in *Proc. of EuroSys*, 2010.
[12] B. Cook, S. Babu, G. Candea, and D. Songyun, "Toward self-healing multitier services," in *Proc. of ICDEW*, 2007.
[13] S. Duan and S. Babu, "Guided problem diagnosis through active learning," in *Proc. of ICAC*, 2008.
[14] G. Jiang, H. Chen, K. Yoshihira, and A. Saxena, "Ranking the importance of alerts for problem determination in large computer systems," in *Proc. of ICAC*, 2009.
[15] A. Whitaker, R. S. Cox, and S. D. Gribble, "Configuration debugging as search: Finding the needle in the haystack," in *Proc. of OSDI*, 2004.
[16] H. Wang, C. Platt, Y. Chen, R. Zhang, and Y. Wang, "Automatic misconfiguration troubleshooting with peerpressure," in *Proc. of OSDI*, 2004.