International Journal of Pure and Applied Mathematics
Volume 116 No. 21 2017, 559-577
ISSN: 1311-8080 (printed version); ISSN: 1314-3395 (on-line version)
url: http://www.ijpam.eu
Special Issue

ijpam.eu

PEER-TO-PEER CONVOLUTED FAULT RECOGNITION TO CONQUER SINGLE-POINT STOPPAGE IN CLOUD SYSTEMS

S.Gokulakrishnan
Research scholar, Faculty of Computing, Sathyabama University, Chennai,Tamil Nadu,India
Assistant Professor. Department of Computer Science and Engineering, Sri Chandrasekharendra Saraswathi Viswa Mahavidyalaya, SCSVMV University Kanchipuram,Tamil Nadu,India

J.M. Gnanasekar
Professor, Venkateswara College of Engineering, Chennai, India

ABSTRACT
Cloud computing provides solutions with unlimited capabilities in such a way to meet the demand of rapidly growing consumerism. Therefore it receives more and more attention from huge players such as backing sectors, educational institutions, government divisions, top-notch enterprises, research organizations etc. At the same time, the unprecedented transformational nature of the Cloud makes it vulnerable to various unpredictable and significant security violations and Byzantine risks. Moreover initial setbacks in detecting byzantine errors often allow it to propagate considerably. Since the byzantine error can simultaneously propagate through various paths it is not feasible to detect all the propagation paths. Many existing solutions fail to detect byzantine faults since they consider monitoring for reactive symptoms rather than capacitating a proactive detection. We consider the security concerns in Cloud as Pure Byzantine problem since it often aims to compromise hypervisors in Cloud initially to act as a decoy to attack the Cloud. Since hypervisors in Cloud exhibit ever growing number of vulnerabilities, they are prone to devastating attacks such as EDoS. Unlike a VM failure, if a hypervisor in Cloud is compromised it allows the error to be propagated throughout the Cloud. Hence in this paper a peer-to-peer hypervisor based verification and validation has been proposed. It performs smart monitoring in the entire set of Hypervisors through promoting intercommunication among them to detect single point failure. The communication involves sending and receiving integrated validation and verification challenge among the Hypervisors at regular interval to improve the possibility of detecting both Byzantine and Pure Byzantine faults. Validation involves checking whether the data generated by the Hypervisors are genuine or not using hash comparison. Whereas Verification involves, checking whether the Hypervisors are performing to the set benchmark or not through performing a bound check for strategically chosen SLA metric.

INTRODUCTION

Cloud computing is considered as the most successful large-scale distributed computing model. The fundamental potential of Cloud involves achieving multilevel virtualization for components such as computing hardware, storage devices, and computer network resources [2]. Since the virtualization has enabled Cloud computing to realize intercommunication and interconnection within various instantaneously deployed virtual machines (VMs) and offered as unified service [2]. High level of virtualization has enabled Cloud computing to seamlessly and dynamically configure multi-terminal, multiplatform, multi-network on-the-fly and offered mostly as pay-as-you-go services at the affordable cost [3][23]. This allows Cloud to expand elastically as well as to meet the high Quality of Service (QoS) requirements [3]. Cloud elasticity has made it capable to be stretched for varying needs and varying levels. Therefore the Cloud deployment is often highly dynamic in nature. Having said that, any Cloud deployment may fall under any of the given categories at any given time they are Private, Public, Hybrid and Community Cloud [1] [5]. Private and Community Clouds are limited to single and similar organizations respectively. Therefore in both the cases Cloud management is considered relatively easy than the Public and Hybrid Clouds which are open and dynamically merged solutions respectively.

Moreover Cloud computing as a service-oriented architecture considers all-inclusive possibilities as anything as service (XaaS) [1]. Where X implies any number of nondeterministic possibilities, when it comes to developing and offering a service using Cloud computing. However the Cloud service providers (CSP) often offer services according to a standard model which considers component inclusion and requirement levels from software to hardware. They are; highest level Software as a Service (SaaS) [4], middle level Platform as a Service (PaaS) [6], and lower level Infrastructure as a Service (IaaS) [6]. The combined use of these three is termed as the SPI model (SaaS, PaaS, IaaS). Iaas may be further branched out in offering services such as storage as a service (SaaS), communications as a service (CaaS), network as a service (NaaS) and monitoring as a service (MaaS). This way with creative ideas the Cloud can be configured and offered as various services and the options are not limited [1][4].

In almost all cases, the choice to move or use Cloud is not restricted solely to the data. Data in the Cloud are just required either to run applications or hosted as part of business processes. However the move towards Cloud is all-inclusive in nature since it is often a step toward reducing the organizational units involved as IT infrastructure. Therefore for all the services the consumer does not manage or control the underlying Cloud infrastructure. Since for all the cases the physical component such as network, servers, operating systems, storage involved are provided by the CSP. Therefore Cloud computing involves functional requirement and non functional requirement. Things to be considered as part of the readiness assessment for functional components from the CSP side includes but not limited to: Connectivity to the Cloud (bandwidth, redundancy), Network security (data encryption), Integration between Cloud and non-Cloud systems, User connectivity (bandwidth to the user end device such as desktop or mobile devices) etc.

Similarly the principal goals of moving to the Cloud from traditional computing often involve but not limited to cost effective, flexible and scalable nature of the services. However to ensure the refined goals the Cloud should be able to react more quickly and inexpensively to changing situations. Therefore Cloud Service Provider (CSP) provisions the customer requirements as agreed upon services with Service Level Agreements (SLAs) [21].

In order to survive rapidly evolving nature of challenges the CSP should be able to expect the unexpected faults because it often struck unpredictably. However to overcome such unexpected challenges are extremely complicated because they are designed to circumvent the Cloud Management. Moreover to counter the challenges it not only requires to create a strong initial set of groundbreaking SLA rules as a proactive measures, reactive measure such as monitoring the performance at regular intervals are considered necessary. Since the Cloud performance metrics are expected to perform consistently from the start till the end. The evolving and evasive nature of the challenges create a huge vacuum when it comes to monitoring the performance metrics. This calls for more research in devising smart algorithms to fill the huge vacuum. It is further made complicated since the Challenges can strike the Cloud from any/many of the levels, such as networks, security, storage, database, software or regulatory initiatives. Moreover Cloud customer operates in an environment that can span geographies, networks, and systems. Therefore for the survival of Cloud businesses for the lack of effective proactive fault avoidance possibility. A contingency plan as fault tolerance for things that can go wrong is often considered to maintain the minimum level of service when such a challenge strikes. However they are limited to VM or node level and don't produce expected results. Therefore in this paper an effective semi reactive detection heuristic that detects the Byzantine fault at hypervisor level rather than at the node level has been proposed.

BACKGROUND

The Cloud services are controlled and monitored using SLA which is unusual for other computing paradigm. Since the user need is effectively taken into the service offering that to on-the-fly through an SLA agreement. It requires through study and analysis upon implications of SLA to further the research effort.

SERVICE LEVEL AGREEMENT (SLA)

SLA involves a contract signed between the customer and the service provider agreeing upon the non-functional requirements of the service specified as Quality of Service (QoS) policy [22]. The Cloud user must determine in advance the terms that will be included in the SLA keeping in mind the fact that strict or complex SLAs could result in higher maintenance cost [21]. However the CSP should anticipate capacity fluctuations otherwise it could result in disastrous service failure. Nevertheless for the customer limited control over selecting, monitoring and configuring the virtual components may be brokered in Cloud Service level agreement (SLA). Since the SLA considers obligations such as service pricing, and penalties in case of agreement violations.
Some of the important QoS requirements that are often negotiated, monitored and documented in the SLA include [3]:

- Availability
- Response time for computing resource requests
- Response time for incidents
- Backup policies
- Data retention and disposal policies and procedures
- Patch management
- Security controls
- Recovery and continuity objectives
- Controls to satisfy legal and compliance requirements

Service level agreement seems to be the backbone of system-level monitoring and Checkpointing. However the SLA is often negotiated by the Cloud broker and every SLA violation is treated differently for different scenarios. For instance the SLA in mission critical applications is more seriously monitored than other applications where certain degree of SLA violation is allowed. The SLA violation may indicate the node (VM) failure, since it fails to cope up with the committed QoS metrics [22]. Whether the node which caused SLA violation is further allowed for processing or replaced with other node is often determined based on the strict or moderate QoS requirements and priorities.

SLA CATEGORIES

SLA can be either dynamic or fixed based on the changes it allows during the operation [8]. The Cloud involves a self-management module often a feature of hypervisor for scheduling the resources based on the SLA agreement [22].

The Static SLA based scheduling does not changes the initially agreed upon terms. It therefore does not impact upon the functional components as well. Such as the number of VMs (nodes), resource allocated to VMs, CPU cycles etc. remain unchanged till the SLA expires. The cost in case of static SLA may not vary much from the determined cost. However it may not be able to cope up with unpredictable behavior of the Cloud components.
Since Cloud services are often prone to load fluctuations and SLA violations during operation. The nature of these fluctuations is unpredictable thus makes the static scheduling less suitable. The Cloud involves a self-management module for scheduling the resources based on the SLA agreement. The dynamic SLA changes the initially agreed upon terms through renegotiation to improvising the performance of the Cloud and to meet the unpredictable changes [32]. It therefore impact the functional components such as the number of VMs (nodes), resource allocated to VMs, CPU cycles if necessary it may also perform VM migration [33]. The cost in case of dynamic SLA if not optimized may become too expensive. Therefore the tradeoff to deploy either static or dynamic SLA involves cost and performance [22].

DYNAMIC LOAD DISTRIBUTION

Cloud computing differ from other distributed computing in offering on-demand scalability. Using dynamic SLA the scalability can be realized by cloning similar tasks onto multiple virtual machines at run-time to meet the workload variations [33]. Load balancer module of hypervisor distributes the workload over the dynamic number of virtual machines at any given time [36]. This helps to accommodate a large number of Cloud users and make the Cloud a multitenant platform. It means that a physical resource can be served to more than one Cloud customer.

CLOUD MANAGEMENT CHALLENGES

BYZANTINE NATURE OF CLOUD FAULTS

When the faulty node ends in a crash it is considered as fail-stop failure, though it is undesirable still it may be detected and replaced [9]. Since there is no processing boundaries with Cloud another node can be simply invoked to fill in the place of the crashed node. However Byzantine failure leaves no such trails, which means that the faulty node generates erroneous data but it appears as a genuine data [9][31]. Thus the Byzantine fault often evades fault detection and makes the fault tolerance almost impossible. However the SLA violation monitoring in case of Byzantine error is troublesome because the node that has faced an error can propagate the error to the other nodes before causing any violation [18]. If this is the case then the error can spread to the successive or arbitrary nodes in a manner to spread out the entire Cloud eventually may drive the entire Cloud system to perform incorrectly without notice. Moreover the byzantine fault can struck public, private, hybrid and community Clouds invariably and can cause the same damage. In mission critical applications such as Cloud run air traffic control, stock exchange etc the impact of byzantine error can be disastrous [7]. Since the Cloud based

huge industries more and more avoid using sensitive data in Cloud infrastructure rather they use non Cloud based infrastructure for hosting and processing the sensible data.

Various attempts were already made to detect the byzantine error and its propagation path [10]. The efforts may not be very successful because the byzantine faults often doesn't cause SLA violations and evade the monitoring. Even if it causes SLA violation it is sporadic in nature and CSP cannot be able to generalize all the occurrences. Moreover initial hiccups in detecting byzantine errors buy it more time to propagate; since the byzantine error can simultaneously propagate through various paths it is not feasible to detect all the propagation paths. Many proposed solutions fail because instead of capacitating a proactive SLA level detection monitoring for reactive violations were considered.

FAULT TOLERANCE

Fault tolerance is the assurance offered by the CSP to continue functionality even if the Cloud is struck with node failures, task failures, hypervisor failure etc. The fault tolerance as a maintenance support is therefore the means to ensure availability and reliability and the main procedure to make the Cloud system more dependable. However to ensure the fault tolerance the CSP should be putting painstaking effort in monitoring the entire set of operating physical and virtual components. The fault tolerance therefore involves steps to detect the failure before or after it happens as well as handling mechanism such as checkpoint and restart [17]. The fault tolerance for unpredictable and undetectable byzantine faults faces various formidable challenges. Since the tolerance highly depends on detection categorizing the detections based on the effectiveness of handling Byzantine faults can offer a better result, it is as follows.

CATEGORIZING FAULT GENERALIZATION

The strength of fault tolerance is found to be bound by the performance of fault generalization. However the fault generalization highly depends upon two metrics they are timing of detection and scale or number of nodes detected. Variation in those two can produce varying results in fault tolerance. Based on that, often the fault tolerance is classified either proactive or reactive however it is only suitable for easily detectable faults such crash fault [5]. However for Byzantine faults the minute variation in detection timing and scale can influence the fault tolerance drastically. Therefore in this paper fundamental categorization for fault generalization using minute variations has been formulated as follows.

Pure Proactive detection: detects the faults before it makes any impact therefore gives no opportunity for Byzantine fault to propagate. Therefore it can help in effective fault tolerance since the fault exhibiting component can be easily replaced with another since the state information such as checkpoint cannot be corrupted. However such detection is only the theoretical possibility because unless the model level solution such as fault avoidance is designed so effective to forbid a byzantine fault possibility [7], it become impossible to realize.

Semi Proactive detection: detects the faults while it is propagating but after making moderate impact. The real-time expectation on measures for monitoring and detection falls within this range. Since it can create the possibility to optimize the fault tolerance in minimizing the damage.

Semi Reactive detection: detects after the fault makes considerable impact. Often results in poor fault tolerance since most of the faulty *nodes* go undetected. The fault tolerance is challenging since the fault is often induced at task level the nodes should have caused at least little erroneous output at various state intervals. Therefore the checkpoint obtained for the previous interval could have made faulty. Therefore Detecting the fault at this range challenges the goal of reliability and dependability of the Cloud to some extend.

Pure Reactive detection: detects fault after it meets its objective. For instance, if the objective is to compromise the entire Cloud system the detection is achieved after it is done so. In this case fault tolerance simply cannot be realized. At this point the obtained checkpoints for previous states must have rendered useless due to the certainty of the existence of the fault.

The main reason for the failure in most of the fault detection and fault tolerance to acheive considerable improvement in Byzantine fault cases is due to considereing tolerance at node level rather than at the hypervisor level.

Significance of Hypervisor

Cloud computing in any case to manage VMs often include various semi automated or fully automated management modules. These modules separate one Cloud from another. The hypervisors such as KVM, XEN,

VMware Vsphere, LXC, UML and MS HyperV are mostly used in the Cloud for managing the VMs [15] [16]. Moreover to install the Cloud any one of the hypervisor is necessary. After installation various Cloud platforms allow to add other hypervisors. This allows CSP to deploy heterogeneous multi-hypervisor based environment yet managed by inherent Cloud management modules.

Table 1: Cloud Mgmt Modules for Various cloud platforms

Cloud Platform	**Mgmt Modules**	**Functionality**	**Hypervisors**
Eucalyptus	Cloud Controller (Solitary & Main module)	Performs tasks such as high-level resource scheduling etc. manages compute, storage, and network resources. offers web interface	KVM, XEN, VMWare
	Walrus	Offers persistent storage to virtual machines	
	Cluster Controller	Manages VM level execution and SLAs per cluster.	
	Storage Controller	manages block volumes and snapshots of VMs within its specific cluster	
	VMware Broker	mediates interactions between the Cluster Controller and VMware	
	Node Controller	hosts the VM instances and manages the virtual network endpoints	
OpenNebula	VM image management	Stores disk images in catalogs	KVM, XEN and VMWare
	Virtual network management	Organizes network catalogs, and using virtual routers interconnect VMs	
	VM template management	Registers VM definitions in the system, to instantiate VM instances later	
	VM instance management	Offers control to VMs, such as migration, stop, resume, cancel, power-off	
OpenStack	Nova	Provision and manage large networks of VMs	KVM, XEN, VMware, LXC Vsphere, UML & MS hyperV
	Swift	Creates redundant, scalable object storage	

Table 1 lists various Cloud management modules for full-featured open source Cloud platforms [15] [16]. The Cloud management modules offer further level of virtualization for Cloud customers. However they still rely on the hypervisors to manage virtual Machines. Since the hypervisors are integrated with the VM management module often they may not be visible. However such module still holds the functionality and behavior of the hypervisors. According to the Table 1 the supervisory modules such as Cluster Controller, VM instance Management, Nova are all the extension of hypervisor to manage the VMs. Therefore the abstraction achieved in the Cloud platform can be simplified as follows.

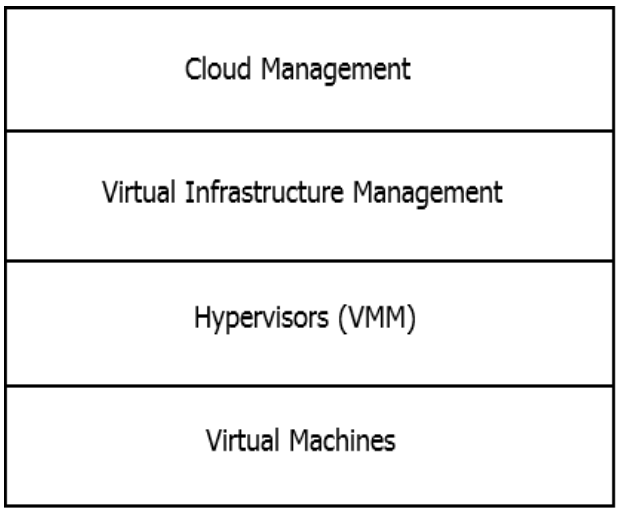

Figure 1 : Level of abstration in Cloud platform

According to figure 1 the components at the Cloud management focus to achieve core features of Cloud such as elasticity, multitenancy, QoS, On demand service etc [3]. Virtual Infrastructure Management (VIM) is often integrated with underlying Hypervisors. Therefore VIM modules assist the Cloud management module by performing supportive tasks such as resource pooling, VM management, managing physical and virtual resources etc. However the name for VIM modules may vary for one Cloud Platform from another still they are integrated to the basic hypervisors and are visible hypervisors. Therefore in this research for simplicity VIM is mentioned as hypervisor.

PROBLEM IDENTIFICATION AND METHODOLOGY

The extreme elasticity has made the Cloud a subject to various extreme challenges, they are as follows.

SINGLE-POINT FAILURE

The Cloud hypervisor is considered to be the main component for a Cloud VM management. Multi-tenancy cases such as, application that scales multiple Clouds or where Cloud adapts more customers dynamically, various Hypervisors are invoked. They are allowed to overlap each other's boundaries to ensure the elasticity [23]. That means, the underlying physical resources are offered to create various virtual resources by various Hypervisors. According to a research the attackers have managed to identify more than 80 vulnerabilities to compromise the single hypervisor [13]. However the list of vulnerabilities grows when considering cases such as add-ons, memory, exposed APIs and login exploits etc [2] [11]. However the Hypervisor as a sole module for management involves the entire feature necessary for monitoring underlying nodes for complete SLA metrics [36]. The SLA metrics is comprehensive in nature to manage them all, includes various features inbuilt in the Hypervisor. Therefore hijacking the hypervisor gives manifold opportunities for the attacker to exploit the Cloud such as hijacking one Hypervisor through another [35]. If the underlying physical resource is same for many Hypervisors the attackers can easily compromise many Hypervisors in no time [35]. Once the Hypervisor is hijacked, the motive for an attacker is to remain undetected and unidentified [31]. Therefore instead of causing more crash faults the attacker thieve through byzantine faults such as inputting junk data or through inducing miniscule node faults so as to cause undetectable errors. Once a node is compromised it feeds faulty output to the other nodes thus causes the error to propagate. Unlike traditional distributed computing the induced errors have the potential to misconfigure the receiving nodes.

More and more security features are enabled to protect the Hypervisor in case of lavish Clouds such as private and community Clouds. Vastly there are many pioneering intrusion detection services that are readily available, they try to keep the Cloud by patching vulnerabilities and detect ongoing attacks using predetermined attack anomaly or signature etc. [20]. Since Cloud is fully Internet based service there are CSP such as Incapsula provides (security as a service) [30]. In spite of all this, according to the recent trend the Cloud attack incidents are growing since there are growing number of vulnerabilities. The motive of the attacker in Cloud also varies from collapsing entire Cloud or corrupting the entire Cloud to smartly maneuvering few elements in Cloud to hide the identity.

Consequently the security concerns in Cloud computing can be interpreted as evolution competence problem. Since the expectation on a deployed security measure is perceived as short-lived and it is expected that the attacker is going to compromise it anyway and in unpredictable ways. Consequently the hypervisor is prone to various kinds of byzantine failure, unlike a node failure if a hypervisor fails the entire cluster may also fails.

EDoS as pure Byzantine fault

Economic Denial of Sustainability (EDoS) as a Cloud based Distributed denial of service attack (DDoS) that thrives to make the Cloud pricing model unsustainable and therefore making it no longer an attractive solution for the customers [28][37]. The traditional DDOS attack tries to deplete the resources of Service Provider to indirectly cause business setback [27]. Whereas the EDOS can directly incur more cost upon Cloud Customers and thus it causes setback for Cloud based business owing to the loss of customers. Due to the dual damage the EDOS can cause for both CSP and customers it has became more dangerous that the traditional DDoS attacks. The attacker through compromising a hypervisor can feeds the nodes under control to process the junk data along with customer data. Moreover it tries to summon as many nodes as possible and dump them with as much task as possible [28]. Each and every task is defined by the attack in such a way to dump the virtual machines or nodes while allowing it to process the genuine customer data. In some cases the genuine user data is processed here and there so as to make the customer believe the data is being processed appropriately. However through doing this the attacker tries to drain as much resources as possible. This is troublesome in Cloud because it follows a pay-per-use pricing model. Therefore the customer is expecting to pay for everything such as for offered processing power, resources etc. Therefore the customers when expecting a cost effective solution the EDoS makes it more expensive solution. This also creates problem between the CSP and the customer. Since the customers is often billed for the agreed upon SLA metrics, the elevated resource usage by the EDoS attack will be reflected in the bill this gives an impression to customers that the CSP is cheating them. However for the CSP since they fail to detect the EDoS attack they consider the usage is genuine. This could cause potential conflict of interest between CSP and Customer thus tests the dependability of the Cloud computing to the Core.

Unlike attacks that involve making the nodes erroneous to cause the byzantine fault, the EDoS thrives through feeding the nodes with junk inputs. This makes the detection even more complicated than the Byzantine problem where though data error is undetectable but testing the nodes with techniques such as hashing algorithm can help to detect the node level or transient fault [14]. Since no such error can be detected in the compromised nodes thus we term it as pure byzantine fault.

CHECKPOINTING PREDICAMENT

Checkpointing is often performed at regular intervals, it involves saving the state of the tasks that are assigned to nodes [11]. If a node fails a new virtual node can be invoked and made to operate with the previous checkpoint of the failed nodes. However this is suitable for fail-stop faults. However for the byzantine faults it has the capability to silently propagate the error to other nodes. As well as, can continue operation through evading the detection thus can make the checkpointed data erroneous as well [11][26].

Moreover Checkpoint/Restart can be implemented at application-level or system-level [24]. In the case of application-level checkpointing, the user has the privilege to manipulate the elements needed to be saved for each tasks and the interval at which it is saved. Hence the user can minimize the application states to be saved for each checkpoint to reduce the checkpoint overheads [24]. However system-level checkpointing performs no modification at application-level. Since no information is available about the application, therefore in most of the cases the checkpoints are all-inclusive in nature and saved for each tasks. This leads to a suboptimal checkpoint performance with exponentially incrementing runtime overhead. Prompting one approach over another however is not possible since most of the cases checkpoint management is left with the service providers. Therefore in spite of all difficulties the system-level checkpointing is expected to function effectively for all the offered services. For this reason, it is important to research the possibilities to support system-level checkpointing with limited overhead and with better efficiency.

METHODOLOGY

The Hypervisor fault is highly compartmentalized since the intrusion that wants the access to the nodes only compromises or shows activities only at the node management module. This makes the detection complicated however if anything that distinguishes the Cloud is its SLA. The SLA involves metrics for almost all operational components and also for non functional component.

Both for dynamic and static SLA the agreement for an application remains the same for all the Cloud Hypervisors as long as there is no renegotiation. When the renegotiation happens, the dynamic SLA is set to be modified at regular interval to match the processing needs of the application. Moreover when re-negotiation happens it is not arbitrary since the Cloud functional components are not visible to the customer. Therefore it is

again distributed among the CSP mutually. Moreover the cause of renegotiation can be positive or negative [22]. In case of positive renegotiations the customer is satisfied with the cost verses performance trade-off and seeks to involve more components. In case of negative renegotiation the customer may not have satisfied with the cost verses performance trade-off and may seek to reduce few components to meet the cost needs. Therefore in case of negative renegotiation the onus is on the entire set of service provides to monitor for service performance in the entire Cloud. Hence for a same service that spans multiple Hypervisors run by multiple service providers the agreed upon SLA needs sensible monitoring for cost incurred.

Moreover in a Cloud like environment where multiple resources from various service providers operate they are monitored separately. This creates various black boxes in one Cloud application. However to compromise the Cloud the attackers exploit this communication gap among the managing units. The attacks such as EDoS attack, normally tries to increase the cost through preying upon vulnerable hypervisor, from there it tries to spread to other hypervisors without notice. Moreover most of the EDoS try to increase the cost significantly without catching the attention of managing unit. Therefore if the cost as metrics doesn't be managed well it causes loss to the credibility. Therefore the proposed work intends to compare and validate the cost based metrics that are available in the Cloud service that is cost per second (c/s) [12]. However the Hypervisor monitors the cost per second but if only a benchmark such as upper cost limit is set for cost it only triggers the detection otherwise it generates no alert for cost variations. Hence in many Cloud cases it is observed only when the variation is too huge to manage that is as pure reactive detection. At that point, tolerating fault becomes complicated and almost renders the Cloud services erroneous and extremely costly.

Conversely rather than performing fault detection and tolerance at node level. The proposed work tries to perform smart monitoring in the entire set of Hypervisors through promoting communication among supervisors to eliminate single point failure. The communication involves sending and receiving integrated validation and verification challenge among the Hypervisors at regular interval. Validation involves checking whether the data generated by the Hypervisors are genuine or not, whereas Verification involves checking whether the Hypervisors are performing to the set benchmark or not.

PROPOSED SEMI PROACTIVE HYPERVISORBASED PEER-TO-PEER BYZANTINE FAULT DETECTION (SPH)

The proposed algorithm requires the hypervisors to share the integrated verification and validation among each other instantaneously for every state interval. It requires a possibility of communication among hypervisors even if they are separated geographically.

HYPERVISOR TO HYPERVISOR COMMUNICATION

The hypervisor is responsible for providing internet connectivity to the virtual nodes [19]. Therefore it has a virtual Network Interface Card (NIC) to connect to the virtual switch. The physical network access is provided as an uplink from the virtual switch to a hypervisor host server's physical NIC that is then connected to a physical Ethernet switch. Therefore each virtual node has a vNIC (virtual Network Interface card) and the nodes together under hypervisor constitute virtual Local Area Network (vLAN). Therefore the hypervisors can connect to each other through vNIC. Moreover there are three types of connection possible between virtual nodes. They are

a) External: if virtual machine or hypervisor connect through physical host machine i.e. through NIC [19]. This is often used by the geographically seperated virtual nodes to connect to each others. Moreover due to n number of vNIC running on single physical NIC it is often overloaded.

b) Internal: if virtual machine or hypervisor connect within the vLAN then they doesn't require physical NIC but it is a within network connection [25].

c) Private: if virtual machines communicate only between each other without communicating to the host operating system [25].

However using the IP address a hypervisor can connect with another privately and therefore it helps to establish the control channel between hypervisor. This way the communication for comparison can be separated from the service level communication since it uses the separate control channel for inter-communication between the hypervisors to promote interoperability.

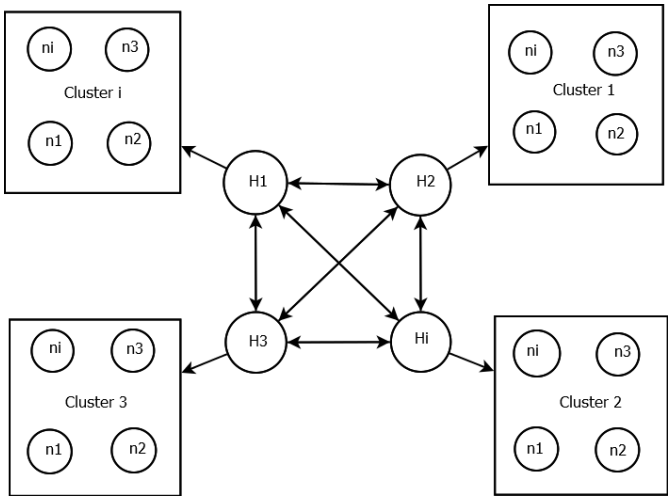

Figure 1: Peer-to-Peer Hypervisor verification and validation Model

Proposed virtual mess topology based peer-to-peer communication and parallel comparison is portrayed as figure 2. The proposed peer-to-peer checking mechanism by utilizing the mess topology allows sharing the challenge among the entire hypervisors in a Cloud application. Then parallel comparison for the shared challenge is performed by every hypervisor. If it is done sequentially or in a random manner it can create gab between successive communications and can create enough room for smart attacker to device means to evade the monitoring before the check completes its round one.

VERIFICATION AND VALIDATION CHALLENGE

The challenge involves Checksum and the SLA metric. However the SLA comparison at i^{th} state interval may or may not operate with the initial SLA agreement, this is expected vastly due to the application of dynamic SLA scheduling and renegotiation possibility. For that reason the proposed work involves identifying key elements in the SLA that marks the impact of renegotiation as well as determines Cloud credibility. The identified key SLA element is cost per second (C/S) since it measures the cost in very short interval and in a wholesome manner as well as specific to each Cloud Controller [12]. Moreover the cost per second has been chosen for SLA metrics since it interprets the performance to the cost incurred, it is what the customer looks for.

The chosen SLA metric C/S for every Cloud hypervisors are then appended with the common message M and then shared between other Cloud Controllers which are part of the same Cloud application.

The receiving Cloud Controller separates the message and the C/S then computes the hash for the message and verifies it with its hash. It validates the performance by comparing SLA C/S with current C/S and it serves as a realistic comparison for cost agreed against cost incurred. If considerable variations are detected, it is communicated to the monitoring authorities as alerts. The algorithm is as follows.

Algorithm 1: Cloud Controller Validation and Verification

Input: predetermined message M with SLA metric c/s

Output: performance alerts

At a fixed time interval Δ
For each Cloud Controller Hi in $\{H_1, H_2, H_3 \ldots H_n\}$
For Predetermined M
Compute Hash h_i with message M and append the SLA metrics 's_i'
Broadcast $M_i\ s_i$
Overall Broadcast ($M_1\ s_1$, $M_2\ s_2$, $M_3\ s_3 \ldots$, $M_n\ s_n$)
H_1 receives array [$M_2\ s_2$, $M_3\ s_3 \ldots$, $M_n\ s_n$]
Similarly for all the Cloud Controller the input is the SLA from others excluding its SLA.
H_2 receives array [$M_1\ s_1$, $M_3\ s_3 \ldots$, $M_n\ s_n$]

H_n receives array [$M_1\ s_1$, $M_2\ s_2$, $M_3\ s_3 \ldots$, $M_{n-1}\ s_{n-1}$]
Operation at H_1

Separate [M_2s_2, M_3 s_3…, M_n s_n] to Message_array [M_2, M_3….M_n] and SLA_array [s_2, s_3…, s_n]
Compute hash_array [h_2, h_3….h_n] for respective Message_array [M_2, M_3….M_n]
Initialize i =0, j=0
FOR i = 0 to hash_array length – 1
if $h_1 \neq h_i$ // This Compare h_1 with [h_2, h_3….h_n]
Store h_i to H_detect [i] //denotes the hash error detection
endif
FOR j = 0 to SLA_array length – 1
if (0 < SLA_array[i] < Upper bound)
// where bound is calculated with acceptable variation x
// Higher bound = SLA +x
// Checks whether [s_2, s_3….s_n] lies within acceptable range
Continue Monitoring:
else
Store s_i to SLA_detect [i] //denotes the unacceptable variation
endif
END FOR
END FOR
if (H_detect length OR SLA_detect length $\neq$ 0)
Unicast arrays H_detect [i] AND SLA_detect [i] to the concerned authority
else
Continue Monitoring
endif

Similarly after completing the operation for the entire set of Cloud Controller the operation for n^{th} Cloud Controller is as follows

Separate [M_1s_1, M_2 s_2…, M_{n-1} s_{n-1}] to Message_array [M_1, M_2….M_{n-1}] and SLA_array [s_1, s_2…, s_{n-1}]
Compute hash_array [h_1, h_2….h_{n-1}] for respective Message_array [M_1, M_2….M_{n-1}]
Initialize i =0, j=0
FOR i = 0 to hash_array length – 1
if $h_n \neq h_i$
Store h_i to H_detect [i] //denotes the hash error detection
endif
FOR j = 0 to SLA_array length – 1
if (0 < SLA_array[i] > Upper bound)
Continue Monitoring:
else
Store s_i to SLA_detect [i] //denotes the unacceptable variation
endif
END FOR
END FOR
if (H_detect length OR SLA_detect $\neq$ 0)
Unicast arrays H_detect [i] AND SLA_detect [i] to the concerned authority
else
Continue Monitoring
endif

If the Cloud Controller observes any deviation it immediately raises alert to the manual supervisor and help the concerned CSP to bring the backup supervisor before the problematic supervisor creates more damage. The proposed algorithm is scalable since it allows any number of Cloud hypervisors to be verified.

The proposed comparison is two way to find out whether any deviation is evident in the SLA agreement or in hash. Moreover it is extendable since it allows checking for any arbitrary number of SLA metrics such as bandwidth utilization, host list, total MIPS, memory utilization etc. Likewise even if the Cloud hypervisors involved are part of single server or single machine it allows benchmark (bound) based monitoring for these elements to ensure Byzantine fault free operation. Since promoting communications among Cloud Controllers in verifying and validating for commonly agreed upon SLA metrics can help to detect the compromise in a semi proactive manner.

Proposed semi proactive detection can make huge difference when it comes to Checkpointing performance. Since the algorithm allows detecting errors at hypervisor level and at the initial stage. This allows CSP to optimize Checkpoint/Restart strategically. Otherwise the CSP have no clue of Cloud hypervisor error and continue Checkpointing for the erroneous cases as normal case. Due to the lack of Byzantine fault generalization, at some point of time the data in the node as well as the data in checkpoint storage may as well get corrupted.

Consequently Peer to Peer Cloud hypervisor based validation and validation can emulate the incremental VM Cloud hypervisor based verification models which involves monitoring for all the required SLA metrics one by one in a Cloud hypervisor after completing the checking the next Cloud hypervisor is checked. Moreover the Cloud Controller thus invoked may are may not be managed by the single CSP. This complicates the scenarios since each CSP uses their own methods for verification and validation. Therefore one CSP that is efficient in detecting certain fault may not be able to detect another. Since the inherent weakness in the incremental model may provide more space for the malicious element to hide its presence through observing the functionalities of the detection method. However the proposed model creates more opportunities for the Cloud hypervisors to act upon to avoid single-point compromise as well as failure.

Mostly occurring faults are due to the faulty communication links in Internet based Cloud Computing. Therefore if the communication links between the Cloud hypervisors are faulty then the message send through the communication link can also becomes faulty. The faulty message will result in faulty checksum; this way using the proposed algorithm apart from generalizing Cloud hypervisor faults the communication link faults can also be detected.

DISCRETE STATE MODELING

The Cloud system hosted cloud application is considered a discrete model to determine the hypervisor behavior at every time intervals. Any cloud application as a discrete system has only finite number of states. Therefore the set of Hypervisors in the Cloud application can also be considered as discrete system since it is summoned to complete the cloud application. The chosen variables C/S and hash are measured every state interval Δ and for finite number of states. These state variables involves observation and testing necessary to detect the faults that cause hypervisor level mode transition from *fail safe state* to *byzantine or pure byzantine state*. During the observation both the discrete component modes and the set of system state variables need to be tracked. The overall system state at time *t* can be described by

$$Y(t)= \Theta (h(t), SLA(t)) \qquad (1)$$

Where $Y(t)$ is the overall system state, $h(t) = [h_1, h_2, h_3, \ldots .h_n]$ is a vector of discrete component modes computed with hash deviation (h') for each hypervisor $H = 1, \ldots .n$ (n: number of hypervisors in the Cloud Applications). Where t is the function of time, it marks the state every constant interval Δ. Assumes a discrete state 'i' from its own set of 'm' states where $h_i = (h_1, h_2, h_3, \ldots. h_n)$, and $SLA_i = (s_1,s_2,s_3, \ldots . ,s_n)$ are the vector variables of system at state 'i'. These continuous variables can be assumed as discrete since the system only involves finite number of states. The vector $SLA(t)$ with $h(t)$ then defines the qualitative values for each system state from a set of *P* possible values. A C/S as SLA variable takes on values from the set of P *{normal, high}*. Similarly, a hash based variable takes on values from the set of possibility *P {accurate, deviation}*. According to algorithm, in case an anomaly is detected the value is transmitted back to the concerned authority otherwise it is not, so the qualitative set can be represented as *P {0, value}*. In both the cases absence of value is marked as normal and presence of value is marked as problem detection. This denotes the binary possibility so the possibility set can be rewritten as P {0, 1} where 0 denotes absence of value or detection and 1 denotes the presence of value or detection.

State Transition

The state transition table and diagram were plotted for a hypervisor level detection at i^{th} state using state variables $\{S_0, S_1, S_2\}$. *The state* S_0 is considered fail safe and hypervisors in this state is considered as error free. The state S_1 may or may not denote problem, it require further state analysis to drive the element in state S_1 to either S_0 or S_2. Where S_2 is considered as acceptor, since the transitions may lead up to the S_2 when the transition reaches the *S2* the hypervisor is confirmed as faulty and requires troubleshooting.

The input to the state machine is a binary set {00, 01, 10, 11}. The binary input is the combination of inputs from both deterministic variables hash and C/S. Hence the input carries the vector that determines the direction within the state machine. The input therefore means 00 – no hash error and no SLA violation, 01 – no hash error and SLA violation, 10 – hash error and no SLA violation and 11 – hash error and SLA violation.

Table 2: state transition table with hash and SLA metric

Present State	Next State 00 01 10 11	Output
S_0	S_0 S_1 S_2 S_2	1
S_1	S_0 S_1 S_2 S_2	1

According to Table II the Output is deterministic because the state transition for given input is deterministic in nature.

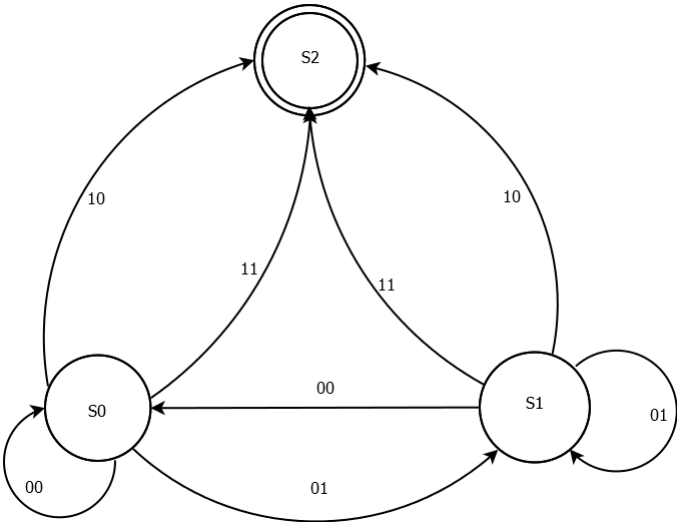

Figure 2: State transition with Hash & C/S

The Table II & figure 3 are constructed with hash and C/S since those are the variables that can detect whether the hypervisor is transitioning into erroneous state or not.

According to figure 2 the hash based validation seems to transition quickly from $S_0 \rightarrow S_2$, and $S_1 \rightarrow S_2$, which implies that the checksum implementation can help to validate the hypervisors effectively. Moreover hash based validation can help to detect the Byzantine error that happens due to faulty hypervisors and the link errors. In both the cases at least a bit in a transmitted message can get corrupted, this indeed causes more variation in the hash due to avalanche effect.

According to figure 3 the transition $S_0 \rightarrow S_1$ and $S_1 \rightarrow S_1$ are incomplete because at state S_1 the detection may not be complete until further analysis. This can happen when the hypervisor is compromised with malicious element and exhibit pure Byzantine fault. As a worst case scenario, if the malicious component is equipped to leave the hash operation intact but still it drives the Cost based metric to intolerable extend. In such cases, therefore the decisive nature of hash may not be helpful. But if the hypervisor remains in state S_1 for more than deterministic times such as 3 or more intervals, then it can be marked as defective, otherwise it is pushing the bound of the incurred cost to unbearable limits. However after transitioning from $S_0 \rightarrow S_1$ if it transition back to S1 $\rightarrow S_0$. This denotes that after exhibiting high C/S in previous state it has recovered from the setback. Therefore generalizing the nodes which show no hash error but shows slightly increased C/S requires consecutive state observations.

The proposed model is designed to be simple since it involves processing small checksum and comparatively simple SLA metric. It is also extendable because it allows additional variables if needed.

The overall state transition is marked with the transition diagram and the table. However the detection confirmation process is more compressive than that, because the confirmation is not with a single state transition rather than by the detection from all the hypervisor. To study this in detail the 2D hypothetical representation using matrix has be explained in the following section.

Hypothetical Matrix Representation

At ever state interval Δ each and every hypervisor processes data from the entire hypervisor space. So as to analyze the Input to Output condition the Verification and Validation challenge in circulation for every state interval Δ has been represented as matrix.

The matrix is always n x n and always a square matrix. Where n is the number of hypervisors, since every hypervisor receives the challenge in row. The row 1 is the challenge received by the H_1 and row 2 is the challenge received by the H_2 and so on until row n is the challenge received by the H_n. The Inputs represented as matrix is given in eqn. (2).

$$\begin{pmatrix} 0 & h2\ s2 & h3\ s3 & \cdots & hn\ sn \\ h1\ s1 & 0 & h3\ s3 & \cdots & hn\ sn \\ h1\ s1 & h2\ s2 & 0 & & hn\ sn \\ & \vdots & & \ddots & \vdots \\ h1\ s1 & h2\ s2 & h3\ s3 & \cdots & 0 \end{pmatrix} \qquad (2)$$

When there is no error or no deviation is observed for hash or C/S then no output is generated and the output is zero. The Resultant Output matrix for fault free case is thus a null matrix as in eqn. (3)

$$\begin{pmatrix} 0 & 0 & 0 & & 0 \\ 0 & 0 & 0 & \cdots & 0 \\ 0 & 0 & 0 & & 0 \\ & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & 0 \end{pmatrix} \qquad (3)$$

If any hypervisor detects the problem all of the hypervisor broad cast the data and the collective data with identified fault can be represented as Problem Matrix as in eqn. (4).

$$\begin{pmatrix} 0 & h2 & h3 & & hn \\ h1\ s1 & 0 & 0 & \cdots & 0 \\ h1\ s1 & 0 & 0 & & 0 \\ & \vdots & & \ddots & \vdots \\ h1\ s1 & 0 & 0 & \cdots & 0 \end{pmatrix} \qquad (4)$$

The matrix denotes the problem in the hypervisor H_1 since except for H_1 all the other elements are validated. Since the hypervisor H_1 is corrupted it can only produce defective hash and while comparison it detects all the other hashes as detective. It may not detect the SLA violation because the received C/S data for other hypervisors are not corrupt and are within the bound. Moreover after detection it sends the false detection to all the concerned hypervisor authorities. Through crosschecking, the hypervisor H_1 can be easily generalized as the erroneous entity not by one but by all the monitoring units.

$$\begin{pmatrix} 0 & 0 & 0 & & 0 \\ s1 & 0 & 0 & \cdots & 0 \\ s1 & 0 & 0 & & 0 \\ & \vdots & & \ddots & \vdots \\ s1 & 0 & 0 & \cdots & 0 \end{pmatrix} \qquad (5)$$

According to the eqn (5) the variation in C/S is spotted with the hypervisor H_1 since all the other hypervisors in unison denotes this to the concerned monitoring unit. This collective alert will put additional pressure upon

monitoring unit to look closely into its working elements either hypervisor or the nodes for troubleshooting. Otherwise such case does not catch the attention of the monitoring unit and can go undetected for a long time.

The eqn (5) is an ideal case, because mostly compromised hypervisors are expected to behave erroneously and exhibit hash errors, since it is the main element of compromise. As a secondary element of compromise the virtual machines or nodes may not necessary exhibit erroneous behavior. Since, in case of pure byzantine fault only the data the nodes process are junk and invalid. Mostly targeted EDoS attacks compromise the hypervisor and try to invade the entire cluster under its control. This therefore requires compromising various modules in hypervisor so it may corrupt the data in hypervisor or the mechanism in the hypervisor. Hence for cases like this, a byzantine fault which corrupts the data or processing in hypervisor is expected. Even if it evade efficiently, it cannot avoid C/S based detection. Since the prime motive of the EDoS attack is to increase the cost incurred to a greater extend.

This way multi party verification can be obtained for a manual supervision. In case if any alert is lost the rest of the alerts will still make it.

RESULT ANALYSIS

Limitations in Detecting Pure Byzantine Faults

Dependability in terms of Cloud computing is multivariate in nature since it should at least be computed with availability, reliability, maintenance support performance and security. The relationship between security and other metrics are tightly-coupled in nature. Therefore depending upon these metrics alone to evaluate performance may be misleading. For instance, a simple manipulation of operational security can mislay other matrices such as availability. Since the smart EDoS is designed to co-occupy the Cloud resources with the genuine processing load it eliminates the detection possibility. This way though creating no availability issues but yet managed to drive the cost to unbearable levels.

However combined monitoring for a metric with security can produce better result as discussed bellow.

Suitability of Chosen Metric

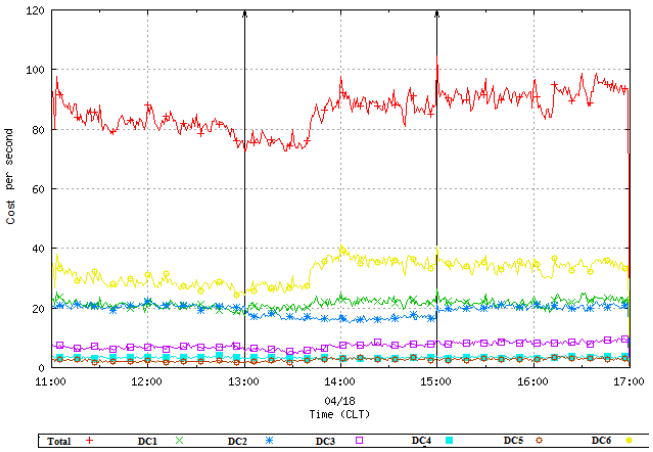

Figure 3: Applicability indication of chosen SLA metric c/s

The figure 3 has been obtained from the Incapsula maintained CSP. The CSP is kept anonymous so not to negatively impact their business. In figure 2 the y-axis 1 unit = 20 cent per second and x-axis 1unit = 1 hour. The cost incurred in cents during the EDoS attack doesn't seems to be shooting up however the percentage of cost promised through SLA is to stay below 15 c/s. The variation in data centers DC1 and DC6 is a definite indication of cost based SLA violation. Looking closely in the obtained dataset the DC6, DC1 and DC2 has been compromised and been entertained for so long. However the variation in the Data centers DC4 exhibits detectable variation even with the naked eye observation. However the EDoS in this case is detected only after the considerable damage has been done. According to figure 2 the C/S based detection is observed to be good fit for earlier pure byzantine fault detection.

Computing the bound

The lower bound can be simply set to zero since the value above the adapted SLA metric is problematic. However if a lower bound is considered it should be optimized. Since the lower bound should be considerable

enough to pick up the rise in cost. For instance after 3 or 4 state intervals if there is no lower bound value observed and suddenly at 5th state interval the value rises to reach the lower bound may further increase at next interval this should be detectable. Moreover the upper bound should be strict to even to detect and alert for drastic variation. Therefore Lower bound for this case can be chosen x-3 and the upper bound x+1, where x = 15c/s. This is specific to this case and it may vary if dollar is in play instead of cent. Therefore it is up to the monitoring unit to choose the bounds appropriately.

Normal case versus Erroneous Case

The obtained data has been categorized as normal case and compromised case as follows.

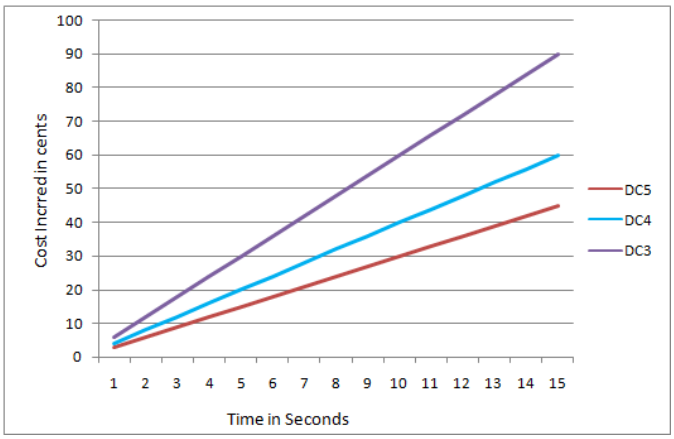

Figure 4: Cost incurred in normal Cloud clusters

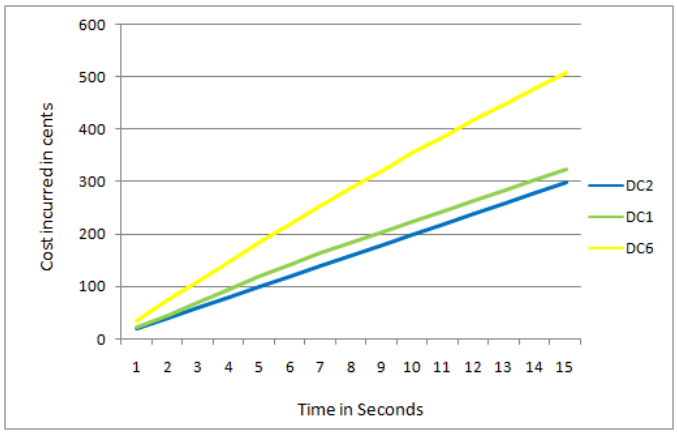

Figure 5: Cost incurred in compromised Cloud clusters

In both the cases x-axis 1 unit = 1 second and for figure 4 y-axis 1 unit = 10 cents and for figure 5 y-axis 1unit =100 cents. Moreover the cost incurred is obtained by adding (current cost + previous cost) in successive manner. The figure 4 is for data centers DC3, DC4, DC5, which are not compromised and operate well within agreed cost that is 15cent/sec. After 15 seconds, according to SLA agreement the cost can go up to 225 cents for each cluster under Data Center. The highest incurred cost for normal case in 15th second is just 90 cents which is caused in DC3. Whereas even the lowest incurred cost in the compromised case is 300 cents that is for DC5. Further interval data is not computed because it is already showing huge variation in some cases. The customer is not paying for the clusters under the data center specifically. Instead the customer is expected to pay collectively. That is at 15th interval according to figures 4 & 5 the payment for the customer is (90+60+45+510+320+300) = 1325. The customer is expecting the collective payment for all the 6 data centers as (6*270) = 1350. Interesting enough, even though few clusters have consumed drastic cost the combined cost at 15th interval has come clean. This therefore is a strategic interval because the customer thinks he is paying for valid data processing and the CSP think they are operating the best they can. Therefore the poor performance of the data centers DC3, DC4 and DC5 which are under EDOS attack are compensated by the best performance of the data centers DC1, DC2 and DC6 thus eliminates the possibility for suspicion or detection.

However in the normal cases the cost is managed very efficiently. Whereas the compromised data center after showing steady variation at every interval at the15th interval it is showing huge variation. Therefore in this

case unless using c/s as a metric at regular interval or at the strategic intervals the variation as in figure 3 may looks normal but it is accumulating additional cost every interval in Stealthy manner.

The success of using hash such as MD5, SHA4 etc in detecting erroneous processing unit is well accomplished. Moreover inducing byzantine error at the real world cloud platforms may cause serious security violations therefore it is not considered for research. However in our previous research we have simulated and evaluated real world dataset for the possibility of using MD5 hash to detect the byzantine error at node level. The result provides evidence that using hash significant and apparent fault generalization capability for byzantine errors at node level can be achieved.

The chosen metrics however can work in tandem, whereas the hash based verification detects Byzantine fault in hypervisor and the C/S based validation detects pure Byzantine fault. Since the attacker cannot gain access to the underlying nodes without compromising the hypervisor. The compromised hypervisor often exhibit Byzantine fault. This helps the attackers to dump junk workloads to properly working nodes this way the problem in node level is improvised to pure Byzantine fault. Therefore the mechanisms that look for faults at node level can be easily fooled by the attacker but the proposed mechanism that tries to look for Byzantine fault in hypervisor and pure Byzantine fault concurrently cannot be fooled.

ANALYSING BYZANTINE GENERALS PROBLEM

A reliable distributed system must be able to cope up with the component failures. A failed component is considered to exhibit a type of behavior that is deviant from the normal behavior which causes the component to spread deviant information to various components of the system. This sort of failure is considered as Byzantine Generals Problem to work out solutions [29].

The Byzantine Generals problem is analogues to the assumed single point failure case. In Byzantine Generals problem there is n number of army generals to head different units. Similarly in the proposed work there is n number of Cloud Controllers to head different Cloud clusters. Moreover the communication between the generals are assumed to be reliable, however in the presented case the transient error may exist but for the simplicity the communication among the Cloud Controllers are considered reliable. However there exist k number of generals as traitors who tries to prevent lieutenants and loyal generals from reaching agreement by feeding them with incorrect information. Further the solution looks at various possibilities by combining lieutenant and commander to come out with best possible solution as for k traitors there should be 3k+1 participants (redundancy) to overcome the Byzantine Generals problem [29][34]. While the Byzantine model can be useful in many computer oriented applications, it will hardly be useful in Cloud computing environments since it is a costly solution.

The performance requirement for fault tolerance is exemplary which demand the Cloud system to hold as many backup VM machines as possible for continues operation. Since nowadays no Cloud platform is limited in terms of switching on the required parallel nodes. The problem is not the availability but the cost in handling redundancy and replication. Simply the challenge for any Cloud System is to achieve high fault tolerance cost effectively no matter what. This can only be achievable if the fault detection so efficient that it demands optimal redundancy and replication [34].

However in the proposed case excluding lieutenants and testing the credibility of generals (Cloud Controller) through signed messages it requires only 2k+1 participants (Cloud Controller) to overcome the Byzantine generals problem with k traitors (faulty Cloud Controllers). The proof is as follows

In the step 1 the loyal Commander sends a value v to all n - 1 Commanders. In step (2), each loyal commander applies his signed command m with n - 1 received messages from generals. Since there is not much time for the traitors to compromise other commanders $n > 2k + m$, where m=1. This implies $n - 1 > 2k + (m - 1)$. Therefore through induction hypothesis since there are at most k traitors, and $n - 1 > 2k + (m - 1) > 2k$. It is safe to say that 2k+1 participants (redundancy) are simply enough to overcome the Byzantine Generals problem using the proposed algorithm.

CONCLUSION AND FUTURE WORK

The proposed algorithm has discussed the means to encourage inter-communication between various Cloud hypervisors to exchange integrated verification and validation challenge among them at regular interval. Since the proposed comparison is two way to find out whether any deviation is evident in the SLA agreement or in hash, if a Cloud hypervisor is unhealthy either due to Byzantine or Pure Byzantine fault then it produces either defective hash or a detectable SLA variation. The suitable SLA metric that can interpret customer's expectation

to the performance has been chosen as Cost per Second (C/S). The proposed validation and verification rather than at the node level it works at the hypervisor level to detect the compromise at earlier stages as semi proactive detection. Therefore it can eliminate the possibility of single point failure. It is extendable since it allows checking for any arbitrary number of SLA metrics such as bandwidth utilization, host list, total MIPS, memory utilization etc. Moreover it can reduce the fault tolerance possibility for Byzantine Generals problem from 3k+1 redundancy to 2k+1 redundancy.

REFERENCES

Bhaskar Prasad Rimal, Erunmi Choi, and Ian Lump.(2009). "A Taxonomy and Survey of Cloud Computing Systems," Fith International Joint Conference on INC, IMS and IDC, pp. 44-51.

Yang Haibo and Tate Mary.(2006). "A Descriptive Literature Review and Classification of Cloud Computing Research", Communications of the Association for Information Systems, Vol. 31, Article 2, 2012.J. P. Martin and L. Alvisi, "Fast Byzantine Consensus," IEEE Transactions on Dependable and Secure Computing, Vol. 3, Iss. 3.

Amid Khatibi Bardsiri, and Seyyed Mohsen Hashemi.(2014). "QoS Metrics for Cloud Computing Services Evaluation", International Journal of Intelligent Systems and Applications, vol.6, no.12, pp.27-33.

Haluk Demirkan and Dursun Delen.(2013). "Leveraging the capabilities of service-oriented decision support systems: Putting analytics and big data in cloud," Elsevier Decision Support Systems, vol.55, pp. 412–421.

Zeeshan Amin, Nisha Sethi, and Harshpreet Singh(2015). "Review on Fault Tolerance Techniques in Cloud Computing," International Journal of Computer Applications, Vol. 116 No. 18.

Khyati Kothari.(2011). "Comparison of several Cloud computing providers," Elixir International Journal of Computer Science and Engineering, vol. 38, pp.4451-4454.

Shigeaki TANIMOTO, Manami HIRAMOTO, Motoi IWASHITA, Hiroyuki SATO, and Atsushi KANAI.(2011). "Risk Management on the Security Problem in Cloud Computing," First ACIS/JNU International Conference on Computers, Networks, Systems, and Industrial Engineering.

Nicola.(2016). "Dynamic SLAs for Clouds," Lecture Notes in Computer Science, pp 34-49, vol 9846.

Shreya Agrawal and Khuzaima Daudjee.(2016). "A Performance Comparison of Algorithms for Byzantine Agreement in Distributed Systems," 12th IEEE European Dependable Computing Conference, pp. 249-260.

Ivona Brandic, Vincent C. Emeakaroha, Michael Maurer, Schahram Dustdar, Sandor Acs, Attila Kertesz, and Gabor Kecskemeti(2010). "LAYSI: A Layered Approach for SLA-Violation Propagation in Self-manageable Cloud Infrastructures", 34th Annual IEEE Computer Software and Applications Conference Workshops.

Cheng Tan, Yubin Xia, Haibo Chen, and Binyu Zang.(2012). "TinyChecker: Transparent Protection of VMs against Hypervisor Failures with Nested Virtualization," IEEE/IFIP 42nd International Conference on Dependable Systems and Networks Workshops (DSN-W), June 2012.

Jacques Sauve, Filipe Marques, Antao Moura, Marcus Sampaio, Joao Jornada, and Eduardo Radziuk.(2005) "SLA Design from a Business Perspective," J. Sch¨onw¨alder and J. Serrat (Eds.): DSOM, LNCS 3775, pp. 73–84.

Diego Perez-Botero, Jakub Szefer and Ruby B. Lee(2013). "Characterizing Hypervisor Vulnerabilities in Cloud Computing Servers," ACM Proceedings of the international workshop on Security in cloud computing, pp.3-10.

Long Wang, Zbigniew Kalbarczyk, Ravishankar K. Iyer, and Arun Iyengar(2010)."Checkpointing Virtual Machines Against Transient Errors," IEEE 16th International On-Line Testing Symposium (IOLTS), July 2010.

Patrícia Takako Endo, Glauco Estácio Gonçalves, Judith Kelner, and Djamel Sadok.(2010). "A Survey on Open-source Cloud Computing Solutions," Brazilian Symposium on Computer Networks and Distributed Systems, vol.71, 2010.

Gregor von Laszewski, Javier Diaz, Fugang Wang, and Geoffrey C. Fox(2012) "Comparison of Multiple Cloud Frameworks," IEEE Fifth International Conference on Cloud Computing, pp. 734-741.

Sheng Di1, Yves Robert, Frédéric Vivien, Derrick Kondo, Cho-Li Wang and Franck Cappello(2013) "Optimization of Cloud Task Processing with Checkpoint-Restart Mechanism", ACM Supercomputing.

Hiep Nguyen, Yongmin Tan, and Xiaohui Gu(2011) "PAL: Propagation-aware Anomaly Localization for Cloud Hosted Distributed Applications", ACM Proceeding Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques, No. 1.

Yi Ren, Ling Liu, Qi Zhang, Qingbo Wu, Jie Yu, Jinzhu Kong, Jianbo Guan, and Huadong Dai(2013). "Residency-Aware Virtual Machine Communication Optimization: Design Choices and Techniques," IEEE Sixth International Conference on Cloud Computing.

Hicham toumi, Ahmed eddaoui and, Mohamed talea(2014) "Cooperative intrusion detection system framework using mobile agents for cloud computing", Journal of Theoretical and Applied Information Technology, Vol.70 No.1.

Saurabh Kumar Garg, Srinivasa K. Gopalaiyengar and Rajkumar Buyya(2011) "SLA-Based Resource Provisioning for Heterogeneous Workloads in a Virtualized Cloud Datacenter," Springer Lecture Notes in Computer Science, vol 7016, pp 371-384.

Andres Garcia-Garcia, Ignacio Blanquer Espert, Vicente Hernandez Garcia(2014). "SLA-Driven Dynamic Cloud Resource Management," Elsevier Future Generation Computer Systems, Vol. 31, pp. 1-11, Feb. 2014.

Fawaz Paraiso, Philippe Merle, and Lionel Seinturier(2016). "soCloud: A service-oriented component-based PaaS for managing portability, provisioning, elasticity, and high availability across multiple clouds", Springer Computing, Vol.98, Iss.5, pp. 539–565.

LIU HaiKun, JIN Hai, LIAO XiaoFei, MA Bo & XU ChengZhong(2012). "VMckpt: lightweight and live virtual machine checkpointing," SCIENCE CHINA Information Sciences, Vol. 55 No. 12, pp. 2865–2880.

Sunay Tripathi, Nicolas Droux, Thirumalai Srinivasan, and Kais Belgaied, "Crossbow: From Hardware Virtualized NICs to Virtualized Networks(2009)." Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures, pp. 53-62.

Tobias Distler and, Rüdiger Kapitz.(2011)."Increasing performance in byzantine fault-tolerant systems with on-demand replica consistency", ACM Proceedings of the sixth conference on Computer systems EuroSys '11, pp. 91-106.

Rajesh, M., and J. M. Gnanasekar. "Path observation-based physical routing protocol for wireless ad hoc networks." International Journal of Wireless and Mobile Computing 11.3 (2016): 244-257.

Rajesh, M., and J. M. Gnanasekar. "Congestion control in heterogeneous wireless ad hoc network using FRCC." Australian Journal of Basic and Applied Sciences 9.7 (2015): 698-702.

Rajesh, M., and J. M. Gnanasekar. "GCCover Heterogeneous Wireless Ad hoc Networks." Journal of Chemical and Pharmaceutical Sciences (2015): 195-200.

Rajesh, M., and J. M. Gnanasekar. "CONGESTION CONTROL USING AODV PROTOCOL SCHEME FOR WIRELESS AD-HOC NETWORK." Advances in Computer Science and Engineering 16.1/2 (2016): 19.

Rajesh, M., and J. M. Gnanasekar. "An optimized congestion control and error management system for OCCEM." International Journal of Advanced Research in IT and Engineering 4.4 (2015): 1-10.

Rajesh, M., and J. M. Gnanasekar. "Constructing Well-Organized Wireless Sensor Networks with Low-Level Identification." World Engineering & Applied Sciences Journal 7.1 (2016).

Rajesh, M. "Traditional Courses Into Online Moving Strategy." The Online Journal of Distance Education and e-Learning 4.4 (2016).

J. Udhayan and T. Hamsapriya.(2011). "Statistical Segregation Method to Minimize the False Detections During DDoS Attacks," International Journal of Network Security, Vol.13, No.3, pp.152-160.

Zubair A. Baig and Farid Binbeshr(2008), "Controlled Virtual Resource Access to Mitigate Economic Denial of Sustainability (EDoS) Attacks against Cloud Infrastructures", International Conference on Cloud Computing and Big Data (CloudCom-Asia).

LESLIE LAMPORT, ROBERT SHOSTAK, and MARSHALL PEASE.(1982) "The Byzantine Generals Problem", ACM Transactions on Programming Languages and Systems, Vol.4 Iss.3, pp. 382-401.

Jos´e Jair Santanna, Roland van Rijswijk-Deij, Rick Hofstede, Anna Sperotto, Mark Wierbosch, Lisandro Zambenedetti Granvillez, and Aiko Pras.(2015). "Booters - An Analysis of DDoS-as-a-Service Attacks," IFIP/IEEE International Symposium on Integrated Network Management (IM), pp.243-251, May 2015.

Lifei Wei, Haojin Zhu, Zhenfu Cao, Xiaolei Dong, Weiwei Jia, Yunlu Chen, and Athanasios V. Vasilakos(2014) "Security and privacy for storage and computation in cloud computing," Elsevier Information Sciences, vol. 258, pp.371–386.

Linlin Wu and Rajkumar Buyya.(2010). "Service Level Agreement (SLA) in Utility Computing Systems," Technical Report Cloud Computing and Distributed Systems Laboratory.

(2012)."Evaluation of gang scheduling performance and cost in a cloud computing system," Springer The Journal of Supercomputing, Vol. 59, Iss. 2, pp. 975–992.

Andreas Haeberlen, Petr Kouznetsov, and Peter Druschel.(2006). "The Case for Byzantine Fault Detection," USENIX Proceedings of the Second conference on Hot topics in system dependability.

Weidong Shi, JongHyuk Lee, Taeweon Suh, Dong Hyuk Woo and Xinwen Zhang(2012) "Architectural Support of Multiple Cloud Controllers over Single Platform for Enhancing Cloud Computing Security," ACM Proceedings of the 9th conference on Computing Frontiers, pp. 75-84.

Pierfrancesco Bellini, Daniele Cenni, and Paolo Nesi(2015). "A Knowledge Base Driven Solution for Smart Cloud Management," Proceedings of the IEEE 8th International Conference on Cloud Computing, pp.1069-1072, July 2015.

S VivinSandar and SudhirShenai(2012). "Economic Denial of Sustainability (EDoS) in Cloud Services using HTTP and XML based DDoS Attacks", International Journal of Computer Applications, Volume 41– No.

P.Delfine Nancy, B.AkoramurthyArthi.J, M. Devi(2017). “A Journey From Virtual Infrastructure To Green Cloud Computing”, International Innovative Research Journal of Engineering and Technology, pp. 21-26, March 2017.