Time-Waved Monitoring and Emergent Self Adaption of Software Components in Open Source Cloud

Conference Paper · September 2015

READS
24

4 authors, including:

Asif Imran
University of Dhaka
18 PUBLICATIONS **26** CITATIONS
SEE PROFILE

Lamisha Rawshan
University of Dhaka
3 PUBLICATIONS **0** CITATIONS
SEE PROFILE

Kazi Sakib
University of Dhaka
40 PUBLICATIONS **52** CITATIONS
SEE PROFILE

Time-Waved Monitoring and Emergent Self Adaption of Software Components in Open Source Cloud

Lamisha Rawshan
bit0311@iit.du.ac.bd

Kazi Sakib
sakib@iit.du.ac.bd

Asif Imran*
asifimran@du.ac.bd

Institute of Information Technology, University of Dhaka
Ramna, Dhaka 1000, Bangladesh

ABSTRACT

Optimized resource utilization and low cost of service has enabled the cloud to become a popular service in today's world. However, rapid scaling, continuous attacks from hackers, dynamic resource provisioning and distributed nature has made it a complex system to manually monitor and manage by system administrators. This paper proposes an effective time-waved framework for monitoring the cloud and reporting undesirable activities with minimum time delay. Next, it presents a mechanism to self-adapt the attacked modules through allocation of healthy ancillary resources. Performance analysis of the proposed framework yields desirable time complexities of 17.0, 26.6, 27.3 and 18.6 seconds for 4 types of attacks tested here. Also, replacing paralyzed cloud virtual machines (vm) with healthy ones requires 8.4 seconds on average, resulting in desirable performance. The experimentation on open source platform show that the proposed schemes enable better monitoring of cloud services.

Keywords

Cloud computing integrity; Cloud forensics; Self-adaptation of cloud components.

1. INTRODUCTION

Modern software executed in Cloud environments has become increasingly dynamic and distributed and those are used by a large collection of clients from different parts of the world through the World Wide Web (WWW). Due to the rapid growth in demand for cloud-based software, the uptime of those needs to be maintained continuously. In this regard, ensuring failure resiliency of the cloud services becomes a significant challenge, mainly due to its distributed characteristics. Additionally, effective monitoring is needed for preserving service integrity and self-adaptation of the software running in the cloud to ensure that the administrator does not have to manually address all the faults. This will play an important role to increase the popularity of the cloud among probable users.

Distributed nature of the cloud components and huge number of data centers hosting the virtual machine vm-instances demand that the integrity of its services is ensured [1]. Also placing vm-instances in such a large platform demands effective monitoring [2]. Research needs to be done on whether it is possible to achieve integrity of the cloud system and its software through time-waved monitoring and self-adaptation of malfunctioning processes. Architectural idea about a time-oriented cloud activity monitoring mechanism and the requirements for effective self-adaptation of complex processes are needed for ensuring integrity of its services. More specifically, the following research questions need to be addressed:

1. *How to achieve time-waved monitoring and real-time feedback for cloud services to ensure its integrity?*

2. *How can securely-adapting critical processes in cloud be achieved to obtain continuous service delivery?*

Traditional applications were presented with self-healing processes to ensure recovery of software in case of failure [3]. Runtime errors were considered as targets that were corrected through the proposed mechanism of analyzing bytecodes to detect errors at runtime, and recovering those processes via manual intervention [4]. Despite the advantages, the approach does not take into account the large number of complex processes that runs and interacts in distributed systems like the cloud. Security issues of cloud computing were taken into account and malicious entities in cloud were detected through signature matching in [5]. However, the authors did not consider the importance of process monitoring and ancillary vm-instances for ensuring integrity and achieving fast recovery of malfunctioning cloud.

This paper proposes a Monitor framework that identifies the pre-specified undesirable cases in cloud and triggers the administration team of experts through dispatching distributed warnings. Here, an interpreter process for monitoring cloud components has been used that produces ($unified2$) binary output files. It takes system behavior as output and writes to a MySQL database, thereby reducing load on the memory. The database is read by the warning process of the Monitor that obtains data for defined events and takes necessary actions according to the pre-specified configuration. Also an automatic retrieval of the Monitor's definitions and configurations is provided for referencing. To achieve reliable notification, $msmtp$ is used

*Corresponding author.

ICEMIS'2015, September 24-26, 2015, Istanbul, Turkey.

DOI: http://dx.doi.org/10.1145/2832987.2833068

which is a lightweight Short Message Transmission Protocol ($SMTP$) client for sending notification messages. A popular mechanism for sending notifications namely TOR, $SOCKS$ proxy and DSN are supported which makes it a robust email client. In addition to the above, the proposed framework detects failures in software modules running in cloud vm-instances using a pre-specified failure-list that is inserted into the cloud controller. Next, the Monitor process running on the same controller node detects faults in the cloud components under observation. The fail-list is read and transferred to the $ReInitiator$ mechanism of the framework that runs in the cloud controller. The difference between the pre-specified failure-list and identified fail-list is specified through exclusive comparison. Ancillary vm-instances are used to replace the failed ones based on the prescription of the fail-list. The prescription of the case-list that has the minimum difference is taken as the solution to the identified problem.

Empirical investigation of performance led to satisfactory results for an e-commerce website used here as a case study that runs in the vm-instances of OpenStack Juno cloud in Ubuntu 14.04 Operating System (OS) platform. The web service, database and network communication were operated for a period of 2 months during the course of this investigation. The proposed Monitor framework was adapted to detect cases of Stealth Port Scans (SPC), Buffer Overflows (BO), Common Gateway Interface (CGI) attacks and Server Message Block (SMB) probes. The time taken for detecting the occurrence of the mentioned attacks to issuing the warning across multiple dashboards and devices of the administrator was recorded and termed as the $Rate\ of\ Reaction$ (RoR) build on the basis of conditions for computations identified in [6]. During the entire duration of experimentation, there were 42 cases of stealth port scans, 16 buffer overflows, 20 CGI attacks and 44 SMB attacks as detected by the proposed Monitor. For the detected cases, the average RoR's for the 4 attacks are 17.0, 26.6, 27.3 and 18.6 seconds per unit respectively, yielding desirable results.

2. RELATED WORK

Recent research has identified the importance of monitoring mechanism and self-adaptation of cloud components for the purpose of ensuring the integrity of its services together with the data stored in it. In this section we highlight some related research efforts on self-adaptation of cloud, its challenges and opportunities for further research.

Lemos et.al has proposed a new logging framework that generates needed data for cloud forensic investigation [3]. Also Cheng et.al effectively identified and analyzed security challenges in vehicular cloud computing [4]. This paper listed some use cases (Debugging and Forensics, Fault monitoring, Troubleshooting, Feature usage, Performance monitoring and Security) that can be profited from analysis of captured log data. They listed the challenges of logging cloud data and gave solution for those.It includes implementation of the proposed framework and it itemizes what they logged. However, the importance of time-stamping and self-adaptation of cloud vm-instances to ensure integrity of the cloud has been addressed to a limited extent.

Optimized workflow distribution and distributed information accountability framework to detect the usage of data by different users in the cloud has been proposed in [7]. Usage information are amalgamated with log files in cloud to ensure accountability of the cloud clients. Object centered approach was used to ensure the requirement of authentication whenever the user data is accessed, thereby triggering automated logging of the actions. The importance of time-stamping then user logging process into the system and recovering failed components of the cloud through effective detection has not been addressed by the authors.

A study on the current state of cloud computing monitoring mechanisms has been studied and directions of improvement are proposed in [8] and [9]. The authors addressed the growing demand for cloud in both the technology and economic domain, identifying the complexities associated with it. Hence, they proposed the need for an effective cloud monitoring scheme that will enable better conduction of its operations from the perspective of resource allocation, virtualization and organization. Xen servers and Nagios were used to experiment the proposed monitoring mechanism. Despite their time contribution, the authors did not consider the necessity to monitor network issues, component failure and cyber-attacks of the cloud. Additionally, they did not experiment with the widely popular virtualization technology of kernel virtual machine (kvm) for the cloud environment.

Rule based mechanism to identify software errors caused by faulty coding have been presented in [10]. Pre-identified fail conditions were considered and failures were detected on the basis of that assumption. Failure characteristics such as buffer overflow and programming bugs have been detected in the code and presented for correction. However, the problem resulting from the sudden closure of processes due to component failures was not taken under consideration.

Self-healing mechanisms of critical sections in software have been put forward in [11]. A list of software modules, connectors and recovering parts are executed in all the nodes. Additionally, the modules present a copy of the fail scenario rules in the elaboration repository as proposed by the authors. Policy based analysis were proposed to produce dependency graphs. Based on those, the self-adaptation rule was selected that takes away the failed instances and recovers those from first state. Due to the policy obligation, the failed components need to restart from the beginning state, hence the necessity of re-initiating modules from the last working state was not taken under considered.

3. COMPONENT ORIENTED MONITORING IN CLOUD

Robust monitoring is a mechanism that keeps a system under surveillance for undesired activities and upon detection, flags those and updates the administrators without delay. Automated Adaptation is the process of autonomously identifying the failures and recovering from those within a short time without any human intervention. For a dynamic system such as the cloud, anomalies can occur due to hardware failure, network unreliability, process failure, error with the virtualization technology or database malfunctioning. Also the cloud computing architecture has become too complex to monitor for unwanted activities manually due to the rapid growth in volume of services. Applications running on the cloud contain resources from multiple virtual machines, hence a large number of physical and virtual machines are involved in the process. In this regard, this paper proposes an effective monitoring mechanism for open source cloud that detects faults in real-time and updates the admin-

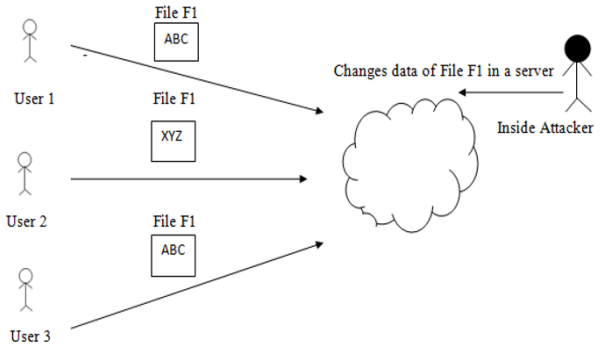

Figure 1: Scenario of service disruption of the cloud components through inside attacker.

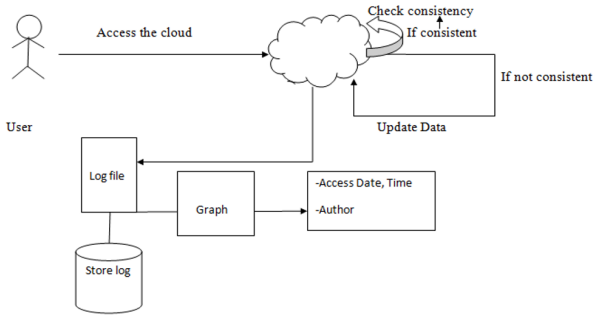

Figure 2: Framework of the proposed solution

istrator. Next, component based self-adaptation procedure proposed here enable rapid solution to the unwanted activity. The proposed framework independently detects and repairs abnormalities on the system with minimal manual intervention. Policy based healing can be initiated without disrupting the normal activities and services provided by a real life cloud platform.

Generally in a dynamic environment such as the cloud a large number of processes execute different virtual machines from various physical machines in order to run an application [12]. Traditionally the system administrator detects the mal-operation and failures manually and takes the necessary steps as per demand. In this regard, maintenance of system administrators for all processes and cloud servers is expensive since a large number of human resources will then be required [3]. Also the manual fixing of errors in the system is time-consuming and late detection of failures can result in significant damage to the cloud. The proposed architecture aims to mitigate this problem through real time monitoring of the cloud environment and self-adapting critical failures in it for quick recovery and service restoration.

As stated in the previous section, self-adaptation is a mechanism that can detect failures in computing systems and recover those autonomously. Component oriented self-adaptation methodology is concerned with the processes which are dependent on each other in a distributed environment such as the cloud. Cloud vm-instances are largely involved in solving the issues of module failures. Failures in an application executed in cloud vm-instances due to the malfunctioning on the processes that are running on those or due to attacks on processes as depicted in Figure 1, the system administrator manually revives the components and fixes the failed cases, resulting in higher service downtime. The objective of this paper is to monitor and detect failed components in the cloud in realtime and revive those automatically within the shortest time delay.

The framework is divided into three modules whereas the entire component oriented self-healing for DSS (Distributed Software System) exhibits three core tasks. The Time-Waver process is also highlighted in Figure 2. The first module of the proposed framework is the Monitor that primarily concentrates on the critical processes in the cloud environment and monitors those for failures.

4. PROPOSED ALGORITHMS FOR SELF-ADAPTION

Initially an empty process list and process information list are generated to contain information regarding the failed components and nodes in the cloud. The Monitor process will constitute of information regarding the cloud system such as vm-instance eating and fixed (Internet protocol) IP addresses, hardware address, port addresses together with process resource allocation information in-terms of bytes. After the Monitor component is triggered at the Cloud controller, it appends both the process id and process information in the initialized buffers. The $GetProcessInfo$ method identifies information regarding memory and processor consumption for the cloud process being currently monitored. This method considers the $process_id$ as an argument and all the critical monitoring data captured by the Monitor as specified above are transmitted to the Time weaver module of the proposed framework.

Algorithm 1 Algorithm running in cloud controller node to monitor the active and failed vm-instances

```
1: procedure MONITOR(P_List, Proc_info)
2:     Process_list ← []
3:     Process_information ← []
4:     while a = num_RequiredProcess do
5:         Monitoring_System(P_id, Proc_info[i],name[j])
6:         If Proc_[i] == InFailedStatProc[j]
7:         Proc_information ← GetProInfo(Proc[j])
8:         return Process is in failed state
9:     end while
10: end procedure
```

Through the proposed mechanism if any redundant vm-instance is required to be deployed then it will be allocated on the basis of the requirement upon analysis of the Message file that stores the output of the status. Afterwards the monitor will pass this information to the time-waved Self-Adapter that will identify the required resources that need to be re-allocated for smooth resumption of the field components. Finally, the self-adaptation will take place as the component will be resumed at a workable state as determined through analysis of the Message file.

$TriggerReciever$ is an additional method, which takes a process id and process information as arguments and the current status of the processed being executed in the vm-instances are returned as shown in Algorithm 1. In case the process status matches the prespecified faulty codes then the

ReInitiator will log this data and send the required healing information to the cloud controller in order to re-launch the faulty instance or allocate new vm-resources.

In case the process is not in a failed state, a message identifying the correct functioning of the process will be generated and the administrator will be notified. Is Faulty method is proposed here that will verify the response from vm-instances whether the cloud based processes are functioning without failure flawlessly or not. The fault identification will be conducted on the basis of predefined threshold value set for the system and also on the error messages from the Operating System (OS). Anomalies that were assumed

Algorithm 2 Algorithm for monitor dependant self-adaptation of components in cloud

procedure SELF-ADAPTOR(P_{Info}, $P_{Collect}$,$count_process$)
  $P_{Info} \leftarrow P_{name}, P_{id}, P_{owner}$
  $M_{bd} \leftarrow message_a, .., message_n$
  $B[x] \leftarrow 0$
  $g \leftarrow 0$
  while $g <= sizeof(Message)$ **do**
    $A[g] = Message$
    $g = g + 1$
  end while
  If $len.Message < len.M_{pr}$, then
  $Message \leftarrow len.Req(T, T_{vm}, S_{vm}, Date)$
  g = g+1;
  return Return system failed information and state to file $Message$
end procedure

in this paper are process crash in cloud computing, runtime-errors, networking connection issues and high resource consumption. Those are the major cases of failures, which are detected by the proposed Monitoring mechanism. Each of the failures and its detection mechanism are discussed here. The process crash occurs when an operation that is non-permitted by the OS is attempted. For example, if a module attempts to obtain access (read or write) of a particular block of system memory which is not permitted for that specific process, a process crash occurs that is popularly known as a segmentation fault. Under certain critical conditions, a process may attempt to execute system instructions by passing unwanted arguments, for example, function on denorms, divide by zero, or passing NULL values can cause process crashing. The Monitor component aims to detect this process failure by identifying the error messages and comparing those with the time-stamps when it happened. If a process does not respond to the Monitor module within specific time then it is consider as a failed process and the required warning are sent to the administrator. Failure of the cloud database and unauthorized faults are identified from the error messages and notified to the mobile devices of the administrator.

5. CASE-STUDY BASED APPROACH

The proposed mechanism can be evaluated via experimentation of the Monitor and Self-Adaptor frameworks identified in Figure 1. The verification and analysis of the performance can be established through implementation of the proposed algorithms in real life open source cloud environment. Here, the test environment consists of OpenStack

Algorithm 3 Algorithm for re-initiating failed vm-instances

procedure REINITIATOR(P_{List}, $Process_{info}$)
  while $i = Proc_{list}$ **do**
    If ($P_{INFO}[I]$ ==IsFaulty($Proc\_status[i]$)), then
    Feedback = ReInitiator($Process_{info}, VM_{info}$)
    Else textbfreturn Message Process is functional
  end while
  $IsFaultyState_{StatProc[k]}$
  return Faulty = Error_Message_vm
  If $StatusandProc_{info} == Faulty$
  Else textbfreturn False
  Proc_list.push(Proc[i])
  $Proc_{info} = Get\_Proc_{info(process[i])}$
  $feedback$ = ReInitiator($stringProc_{info}, vm_{info}$,
  $Policy$)
  return true
end procedure

Juno cloud in CentOs 6.5 platform that hosts web based real life electronic commerce (ecommerce) site running on vm-instances generated using Kernel Virtual Machine (KVM).

The system is monitored for the stated failures using a dashboard where the proposed Monitor mechanism is implemented. The cloud dashboard is used to launch new vm-instances and delete old or failed instances. At the same time it can be used to obtain information about security keys and estimated resource usage. Also the dashboard enables opening and closing of specific ports for communicating with the monitor platform, thereby establishing effective communication between the monitor running in the cloud controller and the vm-instances executing in compute nodes of the cloud.

The monitoring mechanism consists of a web based dashboard identified accessible by the administrators. The case considered here for monitoring is a web based ecommerce web site as described previously. Users can register, post advertisements of different items to sell, browse through catalogues for new products and book those to buy. A web-server will be needed for operation of the e-commerce website. Hence, Nginx is used together with a PostgreSQL database, each of which is configured in individual vm-instances [1]. The PostgreSQL database server is used to store and recover upon query data user registrations, advertisements and product purchasing through the web application.

5.1 Administration of the Monitor Dashboard

The administrator will be able to configure security rules through the implemented dashboard via implementation of the Monitor algorithm proposed here. Next, a new vm from the redundant set of vms is started and assigned for preservation of the previous working state as identified in Algorithm 3. The new vm becomes functional when it is amalgamated with the other instances running the remaining modules of the e-commerce web site used as a test case in this research.

The website used for experiments is loaded in the cloud through a two-layer framework. The first layer consists of the web server and pages. The Domain Name Service (DNS) component is also modified in the vm-instance to obtain dis-integration of the modules. The top tier contains the instances that hold web pages through which users can communicate and look for the goods in the website. The activities at the client includes component execution and actions such as registering, posting advertisements.

The next layer of instances contains $PostgreSQL$ database server that stores data entered by users. Three tables are added initially to the database namely $UserInformation$, $ItemList$ and $PriceOfProducts$. All modules are executed in cloud servers running OpenStack Junno on CentOS 6.5. The vm's are assigned Random Access Memory (RAM) of 1 Giga-Byte (GB) each and 4 virtual Central Processing Units (vCPU). The application layer has 10 GB of storage drive and the database tier has 30 GB of disk storage.

5.2 Interaction amongst distributed components in the cloud

This paper considers the large range of actions such as new user registration, product search and incorporation of new products and reporting activities of the e-commerce web site. Therefore the output obtained from experimentation are applicable to the all e-commerce websites.

The proposed $Monitor$ algorithm is executed in the e-commerce components of the cloud controller to track and diagnose failure of those. More specifically, the service diagnosed by Monitor are grouped as: User Registration, Advertisement posting, Book Item, Network Issues and Intrusion Detection. Under the provided case, the following failures are detected to happen frequently in cloud vminstances and application components running in those.

1. **Web-server failed case**: The web server can e faced with issues of request overflow that may generate from intentional attacks or from a huge number of requests from the users.
2. **Communication failure between Server to Software**: This problem happens when the web-server does not with the SaaS components due to network issues. In case of Service Oriented Architecture (SOA), this problem may occur when one vm component's network process goes down..
3. **Database failure**: PostgreSQL database used here may face configuration problems which occurs during incorrect set up or modification of the process files in the database vm-instance.

The monitor will detect the above mentioned failures and will send a report to the administrator dashboard with minimum time delay. At the same time warning will be sent via Short Message Service (SMS) to the registered mobile phones of the administrators and cloud security team. Also emails provision of the proposed scheme will ensure that the administrators are updated via electronic-mails (E - Mails). As a result, multiple warning mechanisms are incorporated in this framework as discussed above.

6. ANALYSIS OF OBTAINED RESULTS

The experimentation testbed is prepared using the cloud platform as stated in the previous section. The tests are conducted through storing the failed cases in a case-management table identified in Table 1. The first row of the table shows the various component failures that are monitored and the first column highlights the actions of the site visitors intrusion detected by the proposed mechanism. The Monitor is executed in the cloud controller and provides feedback to the administrator in case of any failures of the system. Figure 3 shows graphical results of intrusion detection by the proposed Monitor scheme and issuance of warning.

Table 1: Tabulation of results obtained during experimentation with the four types of attacks

Attack Type	*Detected*	*Time*	*Detection*	*RoR*	*Total*
SPS	42	714	0.7	17	42
BO	16	425.6	0.27	26.6	17
CGI	20	546	0.33	27.3	23
SMB	44	818.4	0.73	18.6	48

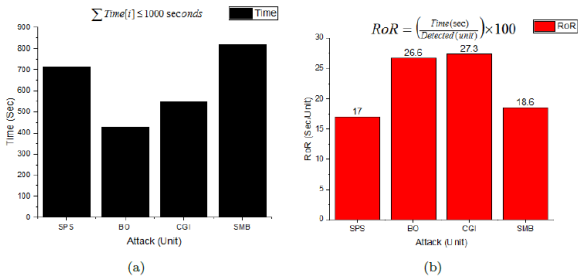

Figure 3: Time and RoR for each attack tested in this research

The experiments consisted of running the web server for a period of 2 months in cloud vm-instance where the visitors carried out a multiple pre-specified actions and Monitor was configured in OpenStack Cloud controller to detect failures. During experiment the undesirable issues were detected with respect to the four criteria of assumed failures described in previous section. Table 1 shows the output of the failed cases in the 4 actions during experimental period. The value of 10 show a failure due to a match in the pre-specified attribute and 00 identifies healthy state of the system.

Upon identification of pre-assumed failure by the $Monitor$, distance between the detected case and pre-specified failure conditions are measured. Exclusive OR is applied in order to detect the difference between the saved failed states and failures of web server operation. Among those, the least difference is determined to detect the closest failure.

The experiment was continued for 2 months and approximately 2000 visitor hits are recorded. In addition, the frequency of accesses by the users of the system are recorded for identifying the number of crashes of system's processes with increasing requests for services from the users. The analysis of the results show that the proneness of the system increases as the number of requests by the users increase, yielding the cloud management team to allocate increased vm-instances and improved monitoring of services. The allocation of more vm's is carried out by scaling up the cloud services horizontally and vertically.

The time absorbed and overhead for incorporation of the proposed framework has been shown in Table 1. The time taken for detection of an attack to updating the administrator for the 4 test attacks cases is depicted in Figure 3. The RoR is shown in the second part of Figure 3 and Figure 4 highlights the comparison of the successful detection and total attack cases. The time overhead for vm-instance assignment is represented graphically in Figure 4. The obtained results highlight the desirable time overhead of the $Monitor$ and $Self - Adapting$ mechanisms proposed here and the capacity of those restore the components in cloud.

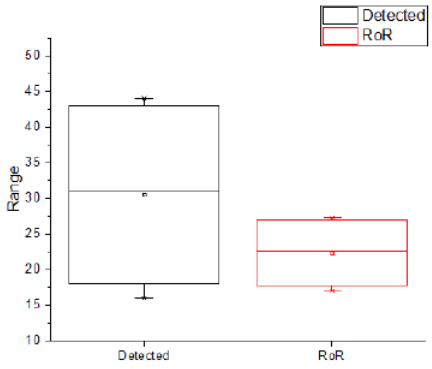

Figure 4: Comparison between number of stacks detected to the total attack cases and also with the RoR

7. CONCLUSION

The paper proposed a time-waved monitoring framework for detecting attacks on complex processes in open source cloud. The target was to identify cases of attack on the cloud system and send distributed warning messages through SMS, E-mail and Dashboard to the administrator with minimal time delay. Afterwards, the proposed mechanism attempted to automatically replace the attacked vm-instances with healthy ones based on a pre-defined fail-list.

It was shown through implementation that the proposed framework was able to detect the four types of attack on the cloud in numerous occasions and successfully sent warnings to the administrator via multiple mechanism. Analysis of the experimental results show that the *RoR* for the four types of attack are found to be 17.0, 26.6, 27.3 and 18.6 seconds per unit respectively, yielding desirable results for *SPS*, *BO*, *CGI* attacks and *SMB* probe type attacks which are tested. Traditional monitoring software are primarily focused on detection of such attacks in a single data center, however it has been shown here that the proposed scheme is applicable to cloud applications. Since cloud has the capability to aid in rapid assignment or replacement of vm-instances in case of crashes, the self-adaptation mechanism harnesses this strength to replace vms that have been affected by malwares.

The proposed framework primarily ensures failure resiliency of the cloud vms and strengthens cloud computing applications through effective monitoring and self-adaptation with minimal time delay for four types of attacks discussed here. Modifying and testing the proposed scheme for detection of attacks like the Distributed Denial of Service (DDoS), TCP=IP spoofing and Operating System fingerprinting provides a scope for conducting future research. The case of extending this research towards detection of virus attacks in the cloud is also an area of future research interest.

8. ACKNOWLEDGMENTS

This research has been supported by the University Grants Commission, Bangladesh under the Grant No-Reg/Admn-3/2015/48743.

9. ADDITIONAL AUTHORS

Md. Sakibur Rahman (Institute of Information Technology, University of Dhaka, email: sakibmas@gmail.com).

10. REFERENCES

[1] Asif Imran, Alim Ul Gias, Rayhanur Rahman, and Kazi Sakib. Provintsec: a provenance cognition blueprint ensuring integrity and security for real life open source cloud. *International Journal of Information Privacy, Security and Integrity*, 1(4):360–380, 2013.

[2] Mazeiar Salehie and Ladan Tahvildari. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 4(2):14, 2009.

[3] Rogério De Lemos, Holger Giese, Hausi A Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M Villegas, Thomas Vogel, et al. Software engineering for self-adaptive systems: A second research roadmap. pages 1–32, 2013.

[4] Shang-Wen Cheng and David Garlan. Stitch: A language for architecture-based self-adaptation. *Journal of Systems and Software*, 85(12):2860–2875, 2012.

[5] Nikolaus Huber, André van Hoorn, Anne Koziolek, Fabian Brosig, and Samuel Kounev. Modeling run-time adaptation at the system architecture level in dynamic service-oriented environments. *Service Oriented Computing and Applications*, 8(1):73–89, 2014.

[6] Asif Imran, Alim Ul Gias, and Kazi Sakib. An empirical investigation of cost-resource optimization for running real-life applications in open source cloud. In *High Performance Computing and Simulation (HPCS), 2012 International Conference on*, pages 718–723. IEEE, 2012.

[7] Shadi Aljawarneh, Christopher Laing, and Paul Vickers. Security policy framework and algorithms for web server content protection. *ACSFâĂŹ07*.

[8] Guido Salvaneschi, Carlo Ghezzi, and Matteo Pradella. Context-oriented programming: A software engineering perspective. *Journal of Systems and Software*, 85(8):1801–1817, 2012.

[9] Hui Song, Gang Huang, Franck Chauvel, Yingfei Xiong, Zhenjiang Hu, Yanchun Sun, and Hong Mei. Supporting runtime software architecture: A bidirectional-transformation-based approach. *Journal of Systems and Software*, 84(5):711–723, 2011.

[10] Carlos Parra, Xavier Blanc, Anthony Cleve, and Laurence Duchien. Unifying design and runtime software adaptation using aspect models. *Science of Computer Programming*, 76(12):1247–1260, 2011.

[11] M Muztaba Fuad, Debzani Deb, and Michael J Oudshoorn. Adding self-healing capabilities into legacy object oriented application. In *ICAS*, volume 6, pages 51–51, 2006.

[12] Elisabetta Di Nitto, Carlo Ghezzi, Andreas Metzger, Mike Papazoglou, and Klaus Pohl. A journey to highly dynamic, self-adaptive service-based applications. *Automated Software Engineering*, 15(3-4):313–341, 2008.