Active-Threaded Algorithms for Provenance Cognition in the Cloud preserving Low Overhead and Fault Tolerance

ASIF IMRAN, EMON KUMAR DEY, KAZI SAKIB
Institute of Information Technology
University of Dhaka, Bangladesh
asif.imran.anik@gmail.com, emonkd@iit.du.ac.bd, sakib@univdhaka.edu

M. ABDULLAH-AL-WADUD
Department of Industrial and Management Engineering
Hankuk University of Foreign Studies, South Korea
wadud@hufs.ac.kr

Abstract: Provenance is the derivation history of information about the origin of data and processes. For a highly dynamic system such as the cloud, provenance must be effectively detected to be used as proves to ensure accountability during digital forensic investigations. This paper proposes active-threaded provenance cognition algorithms that ensure effective and high speed detection of provenance information in the activity layer of the cloud. The algorithms also support encapsulation of the provenance information on specific targets. Performance evaluation of the proposed algorithms reveal mean delay of 8.198 seconds that is below the pre-defined benchmark of 10 seconds. Standard deviation and cumulative frequencies for delays are found to be 1.434 and 45.1% respectively.

Key–Words: Provenance Detection, Cloud Computing, Provenance Encapsulation, Active-Threading

1 Introduction

Provenance is the meta-data that represents information about the operations executed on specific processes at the activity layer. Cloud Computing is the dynamic provisioning of resources from a shared resource pool that stores critical information of the customers and requires provenance of those to achieve accountability. Real life service providers execute mission critical processes and store high-value information in the cloud [1]. Hence provenance needs to be detected for tracking the processes running at the activity layer of the cloud to ensure monitoring of the stored data and aid in digital forensics.

Capturing provenance and encapsulating those on specific processes in a dynamic system such as the cloud involves significant research challenges, since a large number of dynamic processes are involved. Traditional provenance cognition schemes for distributed systems consist of frameworks capturing system log files that has high overheads in terms of execution time. However provenance detection in the cloud must be real time and dynamic, since a large number of physical and virtual machine (vm) instances are involved. Algorithms that capture and bind provenance information to the original data files in real time without causing significant time overhead and loss of fault tolerance have not been considered for the cloud in earlier studies. More precisely, the following research issues need to be addressed.

1. Novel algorithms with active-threading capability that capture provenance in the cloud and encapsulate the data file with the provenance information without causing significant overheads.
2. Empirical investigation to analyze the performance and overheads of the proposed active-threading algorithms at the activity layer and comparing the obtained results with benchmarks.

Existing mechanisms are not suitable for cloud provenance since they cause significant overhead in terms of intermessage transmission time and cause the cloud processes to slow down at the activity layer. *Ko. et al* identified accountability and audability as prime challenges for cloud computing [2]. Through real life scenarios of malicious cloud attacks, the vulnerability of cloud to attacks was shown. However, the research did not focus on how provenance detection can effectively prevent such attacks. Technical and procedural approaches to ensure security of cloud were discussed in [2]. Schemes such as identification of anonymous login and anonymous users through pseudomonas file read were proposed in [3] without performance com-

parison with standard benchmarks.

Based on the research issues identified above, this paper proposes two novel algorithms called ProvCapsule and ProOCal to detect provenance at low overhead in the activity layer. The activity layer deals with all the user inputs and triggers in the cloud. The provenance encapsulation algorithms achieves the functionality without causing significant delay in transmission time by using active-threading modules. The algorithms capture provenance by treating data files as objects and identifies critical information such as access time, number of operations, type of operations, the executing entity, senders and receivers of the data under consideration. The tracked provenance are then stored in separate meta files that are bounded to the original data file. This binding is based on identity number (id-number) allocation to both the provenance and original files and it helps forensic experts track provenance information to specific targets.

The performance of the proposed algorithms have been analyzed in the environment of real life service providers. Provenance at the activity layers of 936 vm-instances were tracked for files ranging from 512-3072 MegaBytes (MB). Next the provenance files encapsulated the original files and out of 936 cases, the average mean time was found to be 8.198 seconds. Standard deviation was determined to be 1.434 which is desirable with an estimated variance 2.056 considering the large number of vm-instances that were used. The results show desirable performance as the overhead incurred in terms of delay because of the encapsulation module of the algorithm is lower than 10 seconds. Cumulative frequencies were obtained to be 45.1% of the files had about 7-8 seconds transmission and global increase in time for provenance, showing the effectiveness of the algorithms.

2   Related Work

The remote storage of data and remote computation is considered as a critical risk for both the cloud service provider and customer [5]. Accountability must be ensured for both customers and service providers of the cloud for ensuring security [5], [6]. In case of a problem, both parties must be made aware about the liabilities and consequences of the issue. Secure auditing, recording and evidence of information stored on the cloud was proposed as probable solutions to the issue. However, the importance of log based analysis of provenance data has been covered to a little extent.

Accountability of cloud services was identified as a complex challenge to achieve by the cloud developers [8]. Through the description of real life scenarios which included storing images on cloud servers, the

real time tracing of operations executed on the cloud were identified. At the same time file-life-cycle logging, file-change logging was identified to be primary challenges [8]. It was described how malicious attackers try to transfer files on the cloud to servers outside using email services. Regulations which hinder the blocking of such transfer were also identified. The study does not provide a provenance detection scheme that identified both file-centric and system-centric operation on the cloud and can be successfully used to identify and block such attacks.

Technical and procedural approaches to ensure privacy is a key goal for software engineers when building a cloud for the production environment [7]. Lower levels of privacy through the use of privacy management tools for system level operations was focused by the authors. Use of pseudonymisation tools to hide the name of real users is proposed. These technologies include anonymous login, pseudonymous identity number and email addresses. Cryptographic mechanisms are used to ensure that the security and integrity of login information. However the use of log-based provenance to aid in data-forensics is discussed to a little extent.

Privacy aware provenance of scientific workflows is stated in terms of data module and privacy policy [9], [10]. The authors raised questions regarding provision of unlimited or fixed number of allowable provenance queries. The provenance information is to be made available to the user assuring the security and privacy of the information. Composite and simple workflows of provenance data were represented as acyclic graphs and then inferences were obtained from those. However, necessity of user defined provenance information and ensuring management of the captured provenance were not considered.

Recent technologies for provenance detection and scientific workflow systems have been identified in [12]. The data are stored in systems using specialized Semantic Web Languages and XML dialects that are stored as files and the tuples stored in relational databases. Workflow processes contained input of the experiments and the output produced by processing the data. The study is only concerned with scientific workflow systems and does not consider provenance detection, storage and management of system-level files or processes involved in the activity layer.

The above discussion amplifies the importance of novel algorithms that can effectively detect provenance and cause low overhead in terms of data transmission time. The importance of parallel active-threading algorithms to preserve the fault tolerance capability of the cloud and encapsulate provenance on the data files is manifold. The need for performance analysis and comparison with defined benchmarks is

also important for the effectiveness of the algorithms.

3 Proposed Algorithms for Provenance Cognition at the Activity Layer

This section proposes algorithms that are necessary to capture the provenance and store it in a provenance file. Next the file is encapsulated as a metafile together with the original data file. The algorithm treats every data file as individual objects and detects provenance for those.

Active-Threaded provenance cognition algorithms ensure that the provenance is detected without having an affect on the fault tolerance ability of the cloud environment through rapid encapsulation. The next subsection describes ProvCapsule, the novel provenance detection and encapsulation algorithm proposed here.

3.1 ProvCapsule: Algorithm that Captures and Binds Provenance to Cloud Data

Initially, each file contains a provenance metafile called F_{PROV} identified by a unique id B_{ID}. Original files identified by F_{ORG} are stored in $DataFile[x]$ array. F_{PROV} are stored in $ProvFile[y]$ array. Each F_{PROV} encapsulates specific F_{ORG} together with its provenance information. Active-Threading is ensured as the algorithms read each F_{ORG} and treat those as individual objects. Next the methodology sets B_{ID} for each F_{ORG} and F_{PROV} to identify those uniquely.

Once the B_{ID} of an object currently in queue matches an object stored in the $DataFile[x]$ array, it will read the input information I_a that is a subset of the original set of inputs I. This process is used to detect and associate provenance information such as date, time, operator and operation of specific $DataFile[i]$ into the $ProvFile[j]$. The data are stored in a separate F_{PROV} file which is then bounded to the original file as a metafile. The F_{ORG} is then transmitted in the cloud due to commands of the activity layer.

3.2 ProvOCal: Algorithm to Map Provenance Information of Memory and Disk Activities

The algorithm for mapping provenance for memory reads and disk writes are discussed here. Two variables g and h are used to compare length of memory read and length of disk writes respectively. After those

Algorithm 1 Provenance Capsule at Cloud Activity Layer

1: **procedure** PROVCAPSULE(a, b)
2:    $B_{ID} \leftarrow key$ and $F_{ORG} \leftarrow DataFile[x]$ and
3:    $F_{PROV} \leftarrow ProvFile[y]$ and
4:    $P_{SB} \leftarrow Loc[z]$
5:    **while** $B_{ID} \neq 0$ **do**
6:       Read inputs in F_{ORG}
7:       Record for every object a in $DataFile[x]$
8:       **while** $a = DataFile[x]$ **do**
9:          $B_{ID} = a.B_{ID}$ and
10:          $F_{ORG}=F_{ORG+1}$
11:          $DataFile[x + 1]=F_{ORG}$
12:          continue
13:       **end while**
14:       Record for every object c in $ProvFile[y]$
15:       If $B_{ID}.F_{PROV}=c.F_{PROV}$
16:       $B_{ID}.F_{PROV} \leftarrow B_{ID}.F_{ORG}$
17:       $B_{ID} \leftarrow B_{ID} + 1$
18:    **end while**
19:    $ProvFile[y]=(Loc[P_{SB}],P_{SB}.Exec,B_{ID})$
20:    **return** $B_{ID}.F_{PROV}$ & $length.DataFile[x]$
21: **end procedure**

---

are recorded, g and h compared with the obtained results. $MInf$ is the variable for storing memory information of F_{ORG} and recording its memory length. $MSum$ adds the length of disjoint specific blocks to calculate the total size of memory required. The result is added to the minimum between one less than the total length or g.

The calculation of disk location and length of the $DataFile[x]$ and $ProvFile[y]$ is similar to the $MemInf$ calculation, with $SBSum$ saving the final value. Finally the difference in the length of $MSum$ and $SBSum$ are calculated on the basis of the matched B_{ID} to determine whether the file is loaded into memory from disk in a compressed format. Hence effective information from the activity layers in terms of memory and disk information can be identified from the algorithm. Next the value of h is incremented as the algorithm returns the disk and memory consumption of the F_{ORG} object at the activity layer of memory management.

4 System Design for ProvCapsule and ProvOCal

The proposed algorithms are implemented in real life environment of Commercial Cloud Service provider

---

Algorithm 2 ProvOCal Algorithm for Provenance Detection in Memory and Disks

---

1: **procedure** PROVOCAL(a, b)
2:     $g \leftarrow 0$ and $h \leftarrow 0$
3:     $MInf = Loc[P_{SB} + 1]$
4:     **while** $g < len.MInf$ **do**
5:         $MSum = MInf$
6:         $S = min(g, len.MInf - 1)$
7:         $MSum = MSum + S$
8:     **end while**
9:     **while** $h < len.SBInfo$ **do**
10:         $SBSum = SBInf$
11:         $T = min(h, len.SBInf - 1)$
12:         $SBSum = SBSum + T$
13:     **end while**
14:     If $len.B_{ID} < len.MemInf$, then
15:         $f \leftarrow len.MemInf - len.B_{ID}$
16:         h = h+1;
17:     **return** Memory and Disk block consumption for provenance capsule
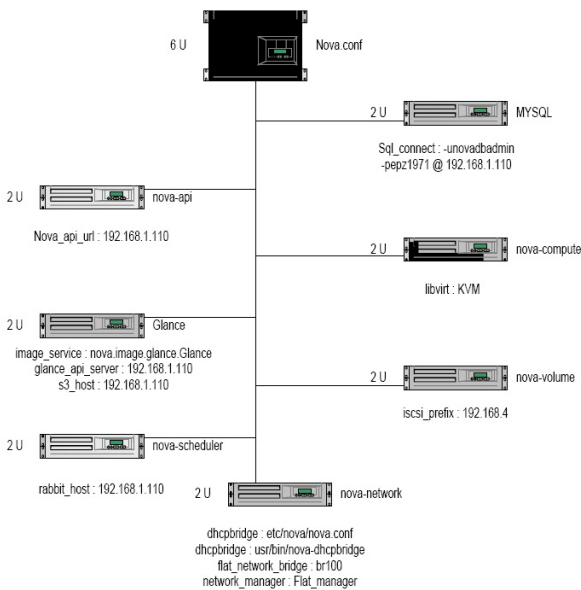18: **end procedure**

Figure 1: Cloud Architecture capable of Provenance Detection

based on OpenStack. The cloud infrastructure has 10 physical nodes and a controller. The algorithms are implemented on the controller node of the cloud. Hence the resource nodes are bound to provide provenance information as directed by the master cloud controller. The processes include operations on files of various sizes ranging from 515 to 3072 Megabytes (MB)used by 960 vm-instances used by the clients

of the cloud service provider. The ProvCapsule and ProvOCal algorithms were implemented in the cloud master controller server.

The vm-instances to which provenance was encapsulated belonged to customers of the cloud provider who ran Business Apps such as inventory management software as Software as a Service (SaaS). Hence the algorithms were tested on real life cloud services.

OpenStack Grizzly Cloud on Ubuntu 12.04 Long Term Service (LTS) server formed the experimental environment. The experiments in the case study included running the applications in virtual machine instances of the cloud. Each instance is allocated 2 GigaBytes of Random Access Memory (RAM) and 2 Central Processing Unit (CPU) cores. Next, the resource consumption and delay of the processes due to encapsulation of the provenance data using the proposed algorithms are identified.

4.1 Sample Scenarios subject to Provenance Detection

In an operating system each event is divided into a series of atomic steps. Each of the steps can be derived from the atomic actions and the kernel system calls. A specific pattern must be followed [14], [4]. This pattern is called the signature of the operation which is necessary for provenance detection since it characterizes and provides behavioral information of the activity of that process.

Analyzing and identifying signatures of text based file operations is a prime goal of provenance detection of file activities in cloud computing environments. Signatures of text oriented file system calls of Linux Operating Systems are listed below.

4.2 Creation of a File

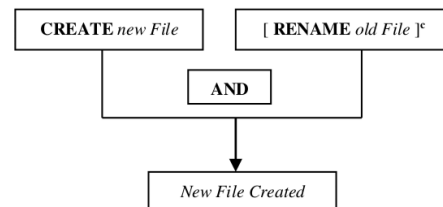- Issue command to **CREATE** text file in a pre-specified process and directory **AND**

Figure 2: Provenance Capture of File Creation and Delete

- The **RENAME** operation is **NOT** executed on the same File in the same process, i.e, [**RENAME** old File]

4.3 Copying a File or its contents to another File

- Issue **COPY** command to the system call, **OR**
- **CREATE** a new File in the same or new directory and **COPY** the contents of the old File and paste them on the new File.

At the Data Layer, the main objective is to analyze the logs which are collected at the System levels [11]. Log files of Cloud Computing provide information that can be used to collect end-to-end system provenance. The provenance information collected in this way will be highly useful to achieve security and trust on behalf of the cloud customers from the cloud service providers.
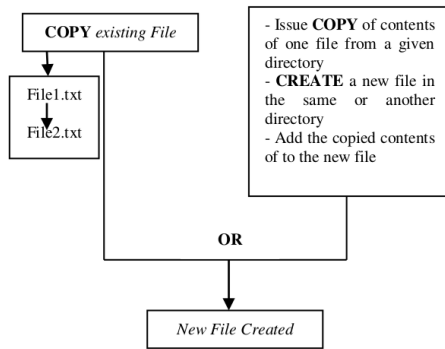

Figure 3: File Copy Operation at the Activity Layer

Based on the analysis of the cloud computing infrastructure and existing provenance models, provenance detection algorithms must be placed forward with an aim of detecting file operations which have possibility of enabling data leakage. ProvCapsule and ProvOCal are such algorithms that aim to drive real time provenance from cloud activity layer without hampering the fault tolerance of the cloud.

The overhead incurred for encapsulating provenance in terms of global delay and message transmission delay needs to be measured for the algorithms proposed here. The results of the algorithms must be compared to established benchmarks to ensure that the overhead incurred does not exceed the pre-defined limit [4]. The following sections identifies the performance of the algorithms in terms of overhead incurred for transmission delay and global delay for encapsulating provenance.

Table 1: Global Delay, Inter-Message Delay (IMT) and Retries of provenance capture

VM-Count	Size	TGD(s)	IMT(s)	Retries
140	512	817	28.9	0
100	1024	867	91.4	0
160	1536	892	90.9	0
162	2048	948	102.7	7
184	2560	1072	169.3	3
190	3072	1118	281.61	7

5 Analysis of Results

The results of overhead calculation for the proposed algorithms are tabulated in Table 1 and Table 2. The number of vm-counts that transferred files of specific sizes ranging from 512-3072 MB are shown in column $VM - Count$. The Total Global Delay TGD is shown for the specific vm-instance counts in seconds. In addition the inter-message transmission delay IMT are also shown in seconds together with the number of retries of provenance encapsulation.

The algorithms were implemented in the cloud controller that hosted 936 vm-instances. Hence there were a finite population of vm-instances X_i such that $X_i = \left(x_1, \quad x_2, \quad \cdots \quad , x_n \right)$. There are a finite population of vm-instances so the mean delay incurred by all those is finite as well. Hence we detect mean, variance and standard deviation for any X_i as,

$$\overline{X_i} = \frac{\sum_{i=1}^{n} X_i}{n} \tag{1}$$

The value can be obtained for a large number of observations n. The variance of the delays in different ranges of Global Delays (TBD) and Inter-Message Time (IMT) is given by,

$$S^2(n) = \frac{\sum_{i=1}^{n}(X_i - \overline{X(n)})^2}{n - 1} \tag{2}$$

The closeness of the variance S_n^2 to mean μ can be determined as $Var(\overline{X(n)})$ is the ratio of total variance and number of occurrences for a given period t_i.

$$STD.dev = \sqrt{\frac{\sum_{i=1}^{n}(X_i - \overline{X(n)})^2}{n(n - 1)}} \tag{3}$$

Table 2 highlights the Mean Time $M.Time$ required for provenance encapsulation for different sized files and different counts of vm-instances. The $M.Time$ is shown to be less than 10 seconds for the
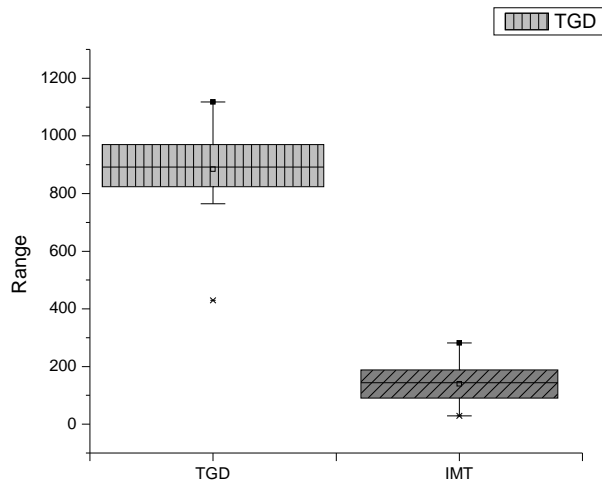
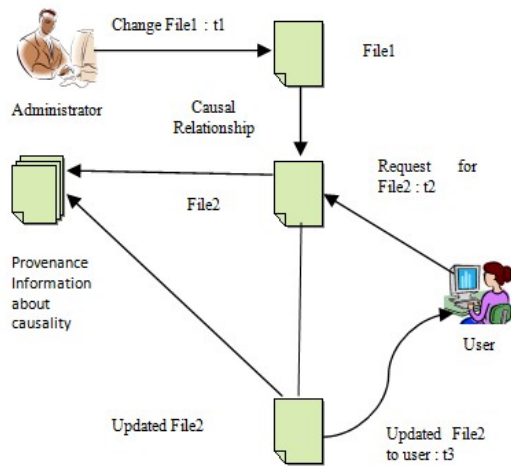Figure 4: Global Delays and Inter-message Delay Distributions

Figure 5: Management of Provenance Information in the Cloud

large number of vm-instances of 936. As identified in [13], the benchmark of tolerable time for provenance encapsulation is 10 seconds for a large system. Taking that value as a benchmark, implementation of Prov-Capsule and ProvOCal algorithms show that the time overhead is below 10 seconds for over 900 instances. The overhead is found to be 8.198 seconds on average which is acceptable. The standard deviation is found to be 1.434 which is desirable.

An important aspect of provenance management regarding data leakage is the causality of information. Information from one process can be causally related to information of another process, hence the atomic
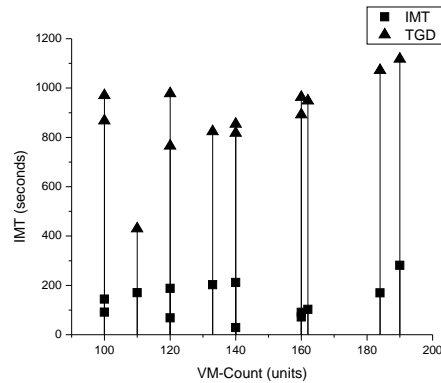

Figure 6: TGD and IMD counts

Table 2: Mean Delay and Standard Deviation of Provenance Encapsulation

Size	M.Cont.	M.Time(s)	M.Delay(s)	STD.Dev
512	5.104	6.042		
100	5.924	8.584		
160	6.368	7.505	8.198	1.434
162	7.011	8.866		
184	8.026	9.990		

actions can be causally combined to evaluate against pre-specified benchmarks and reduce false-positives.

Finally Table 3 shows the delay range and frequency of delay for all the vm-instances. It is seen that 45% of the instances face an average delay of 7-8 seconds. The delay between 9-9.9 seconds is faced by 40% of the instances for encapsulating provenance metafile. Hence the average delay is below 10 seconds for both the algorithms.

The variance and the distribution of the vm-instance classes are shown in Figure 7. The variance is maximum for $M.Con$ whereas it is minimum for $M.Delay$ since the resources allocated in terms of memory and storage for each instance class was different. Hence the benchmark stated in [13] is satisfied.

Table 3: Frequencies and Cumulative Frequencies of different Delay Ranges

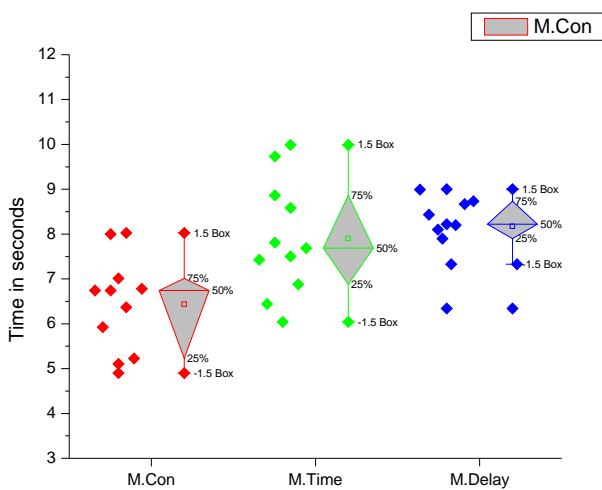| Delay Range	Frequency	Cumulative Freq.(%)
1 - 2	14	1.50
3 - 4	22	2.40
5 - 6	104	11.1
7 - 8	422	45.1
9 - 9.9	374	40.0

Figure 7: Distribution of overhead

6 Conclusion

The paper aimed to ascertain a solution to the bottleneck of capturing provenance data in a widely decentralized system such as cloud computing. Algorithms for detecting provenance and encapsulating those on the specific objects have been proposed in this paper. Compared to traditional provenance detection techniques which function in standalone systems, the proposed active-threaded algorithms are capable of detecting provenance in a virtualized environment.

The performance analysis of the proposed algorithms show that the overhead incurred for capturing provenance using the proposed method is significantly low at 8.198 seconds with retries of 7, 3 and 7 respectively for 936 transactions. The standard deviation of 1.434 and cumulative frequency of 45.1% show desirable performance of the algorithms.

As stated earlier, the proposed mechanism detects provenance for enabling forensic experts to use it in digital forensic investigation. Ensuring performance of the algorithms for preserving scalability of the cloud is a topic of future research interest.

References:

[1] D.J. Pohly, S. McLaughlin, P. McDaniel, and K. Butler. Hi-fi: Collecting high-fidelity whole-system provenance. 2012.

[2] M. Armbrust, A. Fox, R. Griffith, A.D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

[3] A. Imran, A.U. Gias, R. Rahman, A. Seal, T. Rahman, F. Ishraque, and K. Sakib. Cloud-Niagara: A High Availability and Low Overhead Fault Tolerance Middleware for the Cloud. *IC-CIT 2013*, 2013. IEEE. Accepted for Publication

[4] R. Slipetskyy. *Security Issues in OpenStack*. PhD thesis, Norwegian University of Science and Technology, 2011.

[5] P. Storage. The case for content search of vm clouds. 2010.

[6] D. Zissis and D. Lekkas. Addressing cloud computing security issues. *Future Generation Computer Systems*, 28(3):583–592, 2012.

[7] A. Haeberlen. A case for the accountable cloud. *ACM SIGOPS Operating Systems Review*, 44(2):52–57, 2010.

[8] R.K.L. Ko, P. Jagadpramana, M. Mowbray, S. Pearson, M. Kirchberg, Q. Liang, and B.S. Lee. Trustcloud: A framework for accountability and trust in cloud computing. In *Services (SERVICES), 2011 IEEE World Congress on*, pages 584–588. IEEE, 2011.

[9] M. Mowbray, S. Pearson, and Y. Shen. Enhancing privacy in cloud computing via policy-based obfuscation. *The Journal of Supercomputing*, 61(2):267–291, 2012.

[10] R. Lu, X. Lin, X. Liang, and X.S. Shen. Secure provenance: the essential of bread and butter of data forensics in cloud computing. In *Proceedings of the 5th ACM Symposium on Information, Computer and Communications Security*, pages 282–292. ACM, 2010.

[11] A.U. Gias, A. Imran, R. Rahman, and K. Sakib IVRIDIO: Design of a software testing framework to provide Test-first Performance as a service. *2013 Third International Conference on Innovative Computing Technology (INTECH)*, pages 520–525. IEEE, 2013.

[12] I.M. Abbadi. A framework for establishing trust in Cloud provenance. *International Journal of Information Security*, 1(2):1–18, 2012.

[13] H. Park, R. Ikeda, and J. Widom. Ramp: A system for capturing and tracing provenance in mapreduce workflows. year=2011, *Stanford InfoLab, 2011*

[14] S. Akoush, R. Sohan, and A. Hopper. Hadoop-Prov: towards provenance as a first class citizen in MapReduce. In *Proceedings of the 5th USENIX Workshop on the Theory and Practice of Provenance*, pages 11–14. USENIX Association, 2013.