Volume 4, Issue 9, September 2014 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Intrusion Detection System Performance Enhancement Using Dynamic Agent Aggregation and Cloud Based Log Analysis

[1]Manish Kumar[*], [2]Dr. M. Hanumanthappa
[1]Assistant Professor, Dept. of Master of Computer Applications, M.S.Ramaiah Institute of Technology, Bangalore, and Research Scholar, Department of Computers Science and Applications, Bangalore University, Bangalore, India
[2]Prof., Dept. of Computer Science and Applications, Jnana Bharathi Campus, Bangalore University, Bangalore, India

Abstract— ***Intrusion detection is the process of monitoring the events occurring in a computer system or network and analyzing them for signs of possible incidents, which are violations or imminent threats of violation of computer security policies, acceptable use policies, or standard security practices. An intrusion detection system (IDS) monitors network traffic and monitors for suspicious activity and alerts the system or network administrator. It identifies unauthorized use, misuse, and abuse of computer systems by both system insiders and external penetrators. Intrusion detection systems (IDS) are essential components in a secure network environment, allowing for early detection of malicious activities and attacks. By employing information provided by IDS, it is possible to apply appropriate countermeasures and mitigate attacks that would otherwise seriously undermine network security. However, current high volumes of network traffic overwhelm most IDS techniques requiring new approaches that are able to handle huge volume of log and packet analysis while still maintaining high throughput. Hadoop, an open-source computing platform of MapReduce and a distributed file system, has become a popular infrastructure for massive data analytics because it facilitates scalable data processing and storage services on a distributed computing system consisting of commodity hardware. The proposed architecture is able to efficiently handle large volumes of collected data and consequent high processing loads using Hadoop, MapReduce and cloud computing infrastructure. The main focus of the paper is to enhance the throughput and scalability of the IDS Log analysis.***

Keywords— Intrusion Detection System, Hadoop File System, Cloud Computing, MapReduce.

I. INTRODUCTION

Intrusion Detection is the process of monitoring and analyzing the information sources, in order to detect malicious information. It has been an active field of research for over two decades. John Anderson's "Computer Security Threat Monitoring and Surveillance" was published in 1980 and has embarked upon this field. It was one of the earliest and most famous papers in the field. After that in 1987, Dorothy Denning published "An Intrusion Detection Model", provided a methodological framework that inspired many researchers around the world and has laid the groundwork for the early commercial products like Real Secure, Trip Wire, Snort, Shadow, and STAT etc.

Intrusion Detection technology has evolved and emerged as one of the most important security solutions. It has several advantages and it is unique compared to other security tools. As information systems have become more comprehensive and a higher value asset of organizations, intrusion detection systems have been incorporated as elements of operating systems and network. Intrusion detection systems (IDS) have a few basic objectives. Among these objectives are Confidentiality, Integrity, Availability, and Accountability.

Intrusion Detection Systems (IDS) are important mechanisms which play a key role in network security and self-defending networks. Such systems perform automatic detection of intrusion attempts and malicious activities in a network through the analysis of traffic captures and collected data in general. Such data is aggregated, analyzed and compared to a set of rules in order to identify attack signatures, which are traffic patterns present in captured traffic or security logs that are generated by specific types of attacks. In the process of identifying attacks and malicious activities an IDS parses large quantities of data searching for patterns which match the rules stored in its signature database. Such procedure demands high processing power and data storage access velocities in order to be executed efficiently in large networks.

A. Classification of Intrusion Detection Systems

Intrusions can be divided into 6 main types:-

1. Attempted break-ins, which are detected by a typical behaviour profiles or violations of security constraints.
2. Masquerade attacks, which are detected by atypical behaviour profiles or violations of security constraints.
3. Penetration of the security control system, which are detected by monitoring for specific patterns of activity.
4. Leakage, which is detected by atypical use of system resources.
5. Denial of service, which is detected by atypical use of system resources.
6. Malicious use, which is detected by atypical behaviour profiles, violations of security constraints, or use of special privileges.

However, we can divide the techniques of intrusion detection into two main types. IDSs issue security alerts when an intrusion or suspect activity is detected through the analysis of different aspects of collected data (e.g. packet capture files and system logs). Classical intrusion detection systems are based on a set of attack signatures and filtering rules which model the network activity generated by known attacks and intrusion attempts **Error! Reference source not found.**[2][3]. Intrusion detection systems detect malicious activities through basically two approaches: anomaly detection and signature detection.

i. Anomaly Detection

This technique is based on the detection of traffic anomalies. The deviation of the monitored traffic from the normal profile is measured. Various different implementations of this technique have been proposed, based on the metrics used for measuring traffic profile deviation.

Anomaly detection techniques assume that all intrusive activities are necessarily anomalous. This means that if we could establish a "normal activity profile" for a system, we could, in theory, flag all system states varying from the established profile by statistically significant amounts as intrusion attempts. However, if we consider that the set of intrusive activities only intersects the set of anomalous activities instead of being exactly the same, we find a couple of interesting possibilities: (1) Anomalous activities that are not intrusive are flagged as intrusive. (2) Intrusive activities that are not anomalous result in false negatives (events are not flagged intrusive, though they actually are). This is a dangerous problem, and is far more serious than the problem of false positives.

The main issues in anomaly detection systems thus become the selection of threshold levels so that neither of the above 2 problems is unreasonably magnified, and the selection of features to monitor. Anomaly detection systems are also computationally expensive because of the overhead of keeping track of, and possibly updating several system profile metrics. Some systems based on this technique are discussed in Section 4.

ii. Misuse Detection

This technique looks for patterns and signatures of already known attacks in the network traffic. A constantly updated database is usually used to store the signatures of known attacks. The way this technique deals with intrusion detection resembles the way that anti-virus software operates.

The concept behind misuse detection schemes is that there are ways to represent attacks in the form of a pattern or a signature so that even variations of the same attack can be detected. This means that these systems are not unlike virus detection systems -- they can detect many or all known attack patterns, but they are of little use for as yet unknown attack methods. An interesting point to note is that anomaly detection systems try to detect the complement of "bad" behavior. Misuse detection systems try to recognize known "bad" behavior. The main issues in misuse detection systems are how to write a signature that encompasses all possible variations of the pertinent attack, and how to write signatures that do not also match non-intrusive activity. Several methods of misuse detection, including a new pattern matching model are discussed later.

Intrusion detection systems are further can also be classified in two groups, Network Intrusion Detection Systems (NIDS), which are based on data collected directly from the network, and Host Intrusion Detection Systems (HIDS), which are based on data collected from individual hosts. HIDSs are composed basically by software agents which analyze application and operating system logs, file system activities, local databases and other local data sources, reliably identifying local intrusion attempts. Such systems are not affected by switched network environments (which segment traffic flows) and is effective in environments where network packets are encrypted (thwarting usual traffic analysis techniques). However, they demand high processing power overloading the nodes' resources and may be affected by denial-of-service attacks. In face of the growing volume of network traffic and high transmission rates, software based NIDSs present performance issues, not being able to analyses all the captured packets rapidly enough. Some hardware based NIDSs offer the necessary analysis throughput but the cost of such systems is too high in relation to software based alternatives.

From the above, it is clear that as IDS grow in function and evolve in power, they also evolve in complexity. Agents of each new generation of IDS use agents of the previous generation as data sources, applying ever more sophisticated detection algorithms to determine ever more targeted responses. Often, one or more IDS and management system(s) may be deployed by an organization within its own network, with little regard to their neighbors or the global Internet. Just as all individual networks and intranets connect to form "The Internet", so can information from stand-alone internal and perimeter host- and network-based intrusion detection systems be combined to create a distributed Intrusion Detection System (dIDS).

Current IDS technology is increasingly unable to protect the global information infrastructure due to several problems:

i. The existence of single intruder attacks that cannot be detected based on the observations of only a single site.
ii. Coordinated attacks involving multiple attackers that require global scope for assessment.
iii. Normal variations in system behavior and changes in attack behavior that cause false detection and identification.
iv. detection of attack intention and trending is needed for prevention
v. Advances in automated and autonomous attacks, i.e. rapidly spreading worms, require rapid assessment and mitigation, and
vi. The sheer volume of attack notifications received by ISPs and host owners can become overwhelming.
vii. If aggregated attack details are provided to the responsible party, the likelihood of a positive response increases.

II. IDS LOG ANALYSIS USING CLOUD INFRASTRUCTURE

To satisfy demands for the ever-growing network traffic data, IDSs need a deep packet analysis system where the computing and storage resources can be scaled out. Google has shown that search engines can easily scale out with MapReduce and GFS [4][5][8]. MapReduce allows users to harness tens of thousands of commodity machines in parallel to process massive amounts of data in a distributed manner simply defining map and reduce functions. Apache Hadoop [4], sponsored by Yahoo!, is an open-source distributed computing framework implemented in Java to provide with MapReduce as the programming model and the Hadoop Distributed File System (HDFS) as its distributed file system. Hadoop offers fault-tolerant computing and storage mechanisms for the large-scale cluster environment.

A. Cloud Computing and The MapReduce Framework

In this section we discuss the basic concepts of cloud computing and the MapReduce framework, introducing the architecture of the MapReduce implementation provided by the Hadoop project. The constant and rapid growth in the volume of data and communications in current networks and information systems raises the need for efficient scalable data storage and processing techniques. In order to address this issue, a new paradigm commonly called cloud computing was introduced. The main characteristic of this paradigm is the decentralization of data processing and storage through clustered environments that seamlessly scale to fit the constantly increasing demand, offering high performance and achieving efficient response times.

Usually a cloud computing environment is composed by a data processing cluster coupled with a storage area network to provide easily scalable resources. The users and applications access this environment seamlessly and transparently through standard API frontends, eliminating the need for knowledge of the specific network infrastructure and underlying services. Such architectures enable fast development and provisioning of large scale applications that handle a high number of requests or large quantities of data. Applications running on a cloud environment are able to scale transparently and automatically by simply adding more cloud resources, which are handled by the corresponding API.

A cloud application can be easily developed by utilizing the proper frontend API calls to perform the desired actions on the cloud environment (e.g. store and retrieve files or sort data). After an application is functional, it can be transparently scaled by adding more cloud storage or processing resources. Hence, cloud applications can potentially scale to handle any quantity of data. These characteristics make such platforms ideal for constructing efficient and scalable distributed intrusion detection data analysis systems.

This approach makes applications extremely flexible, since MapReduce allows for dynamic configuration of the quantity of maps and reduces. In other words, it is possible to configure the granularity of the data splitting process, thus adjusting the size of transitory key/value pairs that are distributed among the processing nodes in the cloud.

B. The Hadoop MapReduce Architecture

Hadoop is an open-source distributed framework written in Java programming language under Apache License[6][4][7][9]. It supports functions for processing huge size data. Hadoop system is constructed with a master server and multiple slave nodes as shown in Fig. 1. Hadoop has two logical modules: HDFS (Hadoop File System) and MapReduce. HDFS manages duplicated blocks for data and meta-data in order to tolerate fault in a node. A file is composed of 64Kbytes blocks and a block is duplicated to three replicas in general. When a block is created, the first replica of the block is written in the DataNode where the block is created. The second replica is written in the different DataNode in the different rack. Rack is a group of DataNodes that shares the same network switch. The store location for the replica is chosen randomly in order to increase concurrency. Thus a file is distributed to multiple nodes which can be accessed concurrently. Namenode process in the master server stores metadata and locations of the replicas.

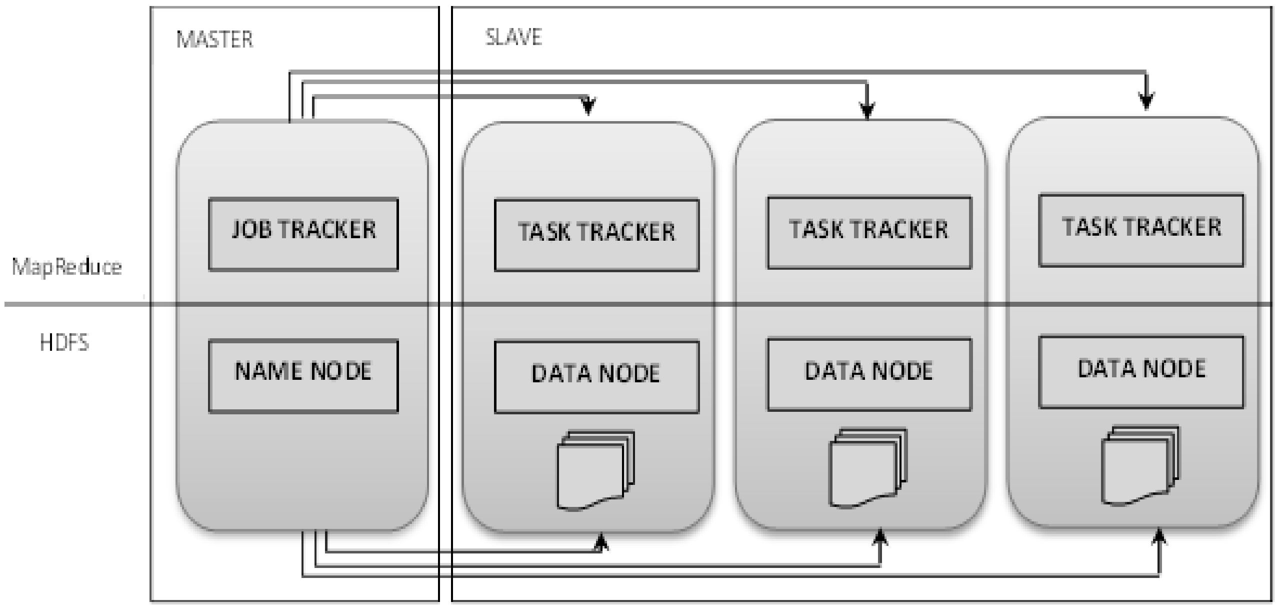

Fig 1: Hadoop Cluster Structure

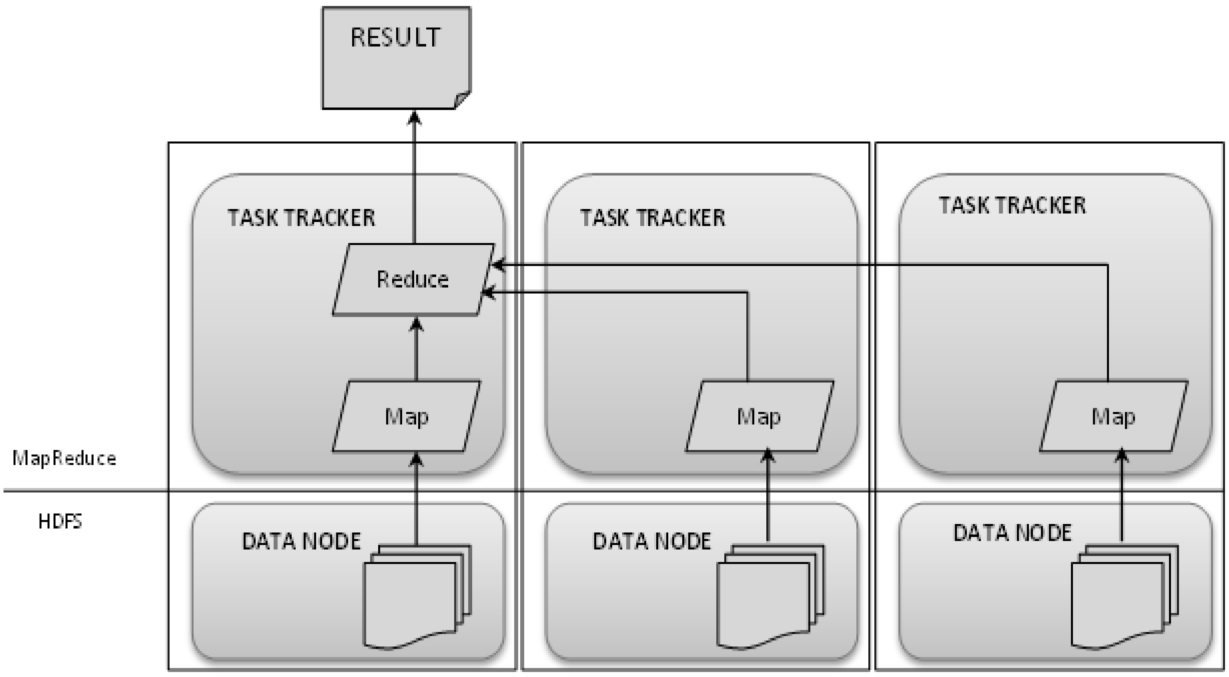

Fig 2:- Hadoop MapReduce Function

MapReduce component in Hadoop controls job processing in distributed manner. JobTracker process in the master server allocates jobs to TaskTrackers in slave nodes and merges results received from the TaskTrackers. When a task process a file in HDFS, each block of the file can be processed by an allocated task in DataNode in each slave node. In order to process the file, jobs are allocated to the slave nodes using map function provided by Hodoop. Result of the jobs are transmitted to the node where reduce function runs as shown in Fig. 2. Reduce function arranges the result and generates a result file. The number of nodes that run reduce function is determined by the number of blocks in the file and the number of reduce functions is decided by user parameter.

III. INTEGRATING IDS INTO CLOUD COMPUTING

We are implementing the IDS log analysis using the cloud architecture with the help of Hadoop and MapReduce. The log file is produced by IDS. Regular Parser, Analysis Procedure, Data Mapper, Data Reducer are processes component of our cloud based IDS log analysis architecture as shown in Fig 3. Meta file is a file transferred into distributed file system. All of metadata are intermediate product of Job Dispatcher between Data Mapper and Data Reducer. At the beginning of procedure, alert generator collects malicious packets and stores information into a log file. The first component, Regular Parser, extracts the essential information and discards useless data then parses as a regular form. Next step, system would transfers the metafile to distributed file system which splits metafile spread every node. Job Dispatcher launches Data Mapper and assigns jobs to every node. The major work of Data Reducer is to reduce the redundancy and merge information. The reduce rule is: if any of two alerts that destination IP and signature are respectively the same, the two would integrate as one and other attributes should be merged. For example, there are six alerts in metadata shown as table 1, and table 2 is the result. I1 and I2 are the same approach because that attacker does the method twice at different time. After analysis job worked, I1 and I2 reduced as R1. The different attribute, I3's b and I4's c, should be merged as R2's {b, c} because they have the same attributes, destination IP and signature. The example is that there are two hosts using the same method to attack one target. Any of the major attributes, signature and destination IP, are different, reduce job would not merge. For example, I5 and I6 are respectively on behalf of R3 and R4.

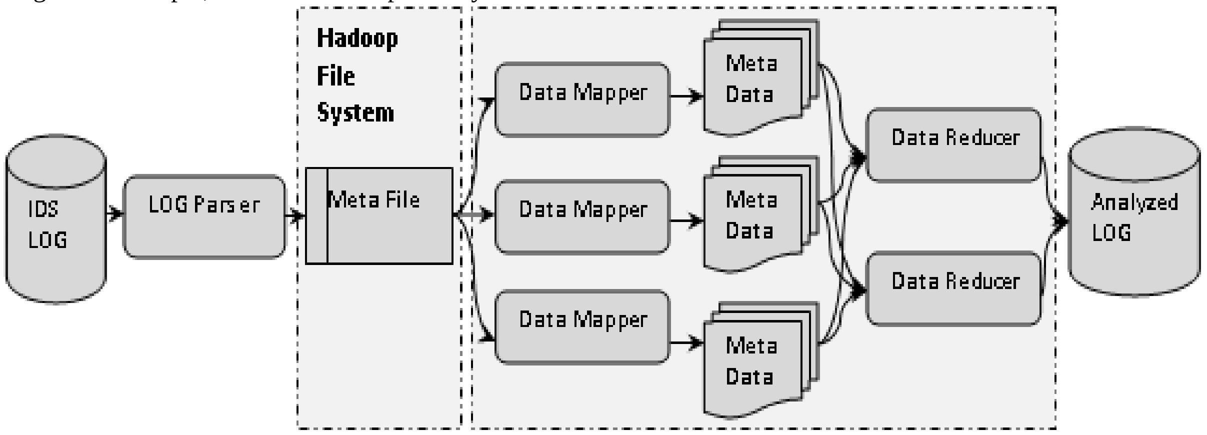

Fig 3:- IDS Log Analysis Using Hadoop MapReduce

Table I: Initial Log Data Sample

ID	IP_Dst	Signature	Others
I1	IP 1	X	{a}, {b},…
I2	IP 1	X	{a}, {b},…
I3	IP 2	X	{a}, {b},…
I4	IP 2	X	{a}, {c},…
I5	IP 3	X	{a}, {b},…
I6	IP 3	Y	{a}, {b},…

Table II: Log Data MapReduce

ID	IP_Dst	Signature	Others
R1	IP 1	X	{a}, {b},…
R2	IP 2	X	{a}, {b,c},…
R3	IP 3	X	{a}, {b},…
R4	IP 3	Y	{a}, {b},…

IV. DYNAMIC AGGREGATION AND SENSOR SELECTION

In our proposed architecture, IDS LOG can be collected from multiple sensors or agent. In this section, we present a trust-based algorithm which dynamically determines the best aggregation agent and also the optimal number of malicious or legitimate behavior necessary for the reliable identification of the best aggregation agent, while taking into account the: (i) past effectiveness of the individual aggregation agents and (ii) number of aggregation agents and the perceived differences in their effectiveness. We decided to use a trust-based approach for evaluating the aggregation agents, because it not only eliminates the noise in the background traffic and randomness of the challenge selection process, but accounts for the fact that attackers might try to manipulate the system by inserting misleading traffic flows. An attacker could insert fabricated flows [10] hoping they would cause the system to select an aggregation agent that is less sensitive to the threat the attacker actually intends to realize. When using trust, one tries to avoid this manipulation by dynamically adapting to more recent actions of an attacker.

For each time step $i \in N$, the algorithm proceeds as follows:

i) Let each aggregation agent classify a set of known instances of malicious or legitimate behavior from different attack classes and selected legitimate known instances of malicious or legitimate behavior.
ii) Update the trust value of each aggregation agent, based on its performance on the known instances of malicious or legitimate behavior in time step i.
iii) Accept the output of the aggregation agent with the highest trust value as classification of the remaining events of time step i.

Known instances of malicious or legitimate behaviour detection and aggregation agents in each time step *i* with the sets of flows for which we already know the actual class, i.e. whether they are malicious or legitimate. So, we challenge an aggregation agent α with a set of malicious events, belonging to *K* attack classes and a set of legitimate events drawn from a single class. With respect to each class of attacks k, the performance of the agent is described by a mean and a standard deviation: $(\bar{x}^k, \sigma_x^k)$ for the set of malicious challenges and $(\bar{y}, \sigma_x)$ for the set of legitimate challenges. Both means lie in the interval [0, 1], and $\bar{x}^k$ close to 0 and $\bar{y}$ close to 1 signify accurate classifications of the agent respectively.

V. EXPERIMENTAL RESULTS

In order to measure performance of the proposed system, we constructed a cluster computing system composed of 5 systems of Pentium Core i5 processor. A master node that has a NameNode and a TaskTracker has 4 Gbytes RAM. Apache Hadoop 2.1.0 is installed on Ubuntu 13.04 operating system. Java development package JDK 1.7.0 is used. The nodes are connected by 100 Mbps Ethernet network switch. Block size in HDFS is set to 64Mbytes and the number of duplicate is set to 2.

We captured the SNORT IDS log report of simulated attack. The total size of the files is about 4 Gbytes. Table 3 shows execution time of Hadoop based IDS log analysis system in comparison with a single computer system without Hadoop is used, and when the proposed distributed Snort system with various number of slave nodes is used. Since there is no overhead on resource or task management in a single computer system, single system has better performance than the proposed system with a single slave node does. As the number of nodes increases in the proposed system, execution time decreases. From two nodes system, the performance of the proposed system starts increasing compare to that of the single system. When the number of nodes in the system is 5, performance of the system is about 2.76 times better than that of the single system in cluster. Fig.4 shows the decrease in execution time required for log analysis as we increase the number of node and Fig. 5 shows that Gradient of speed up is almost linear, which means that the proposed system is highly scalable. The table also shows that how We also found that the as the number of nodes increased in cluster the speed up increased compare to the execution time taken in to a normal IDS system without having any kind of

Table III: Comparison of Execution Time and Speed Up

Number of Nodes	**Execution Time (sec)**	**Speed Up By Factor**
Single System	303.3	
1	475.3	0.64
2	236.9	1.28
3	165.9	1.82
4	126.5	2.39
5	109.5	2.76

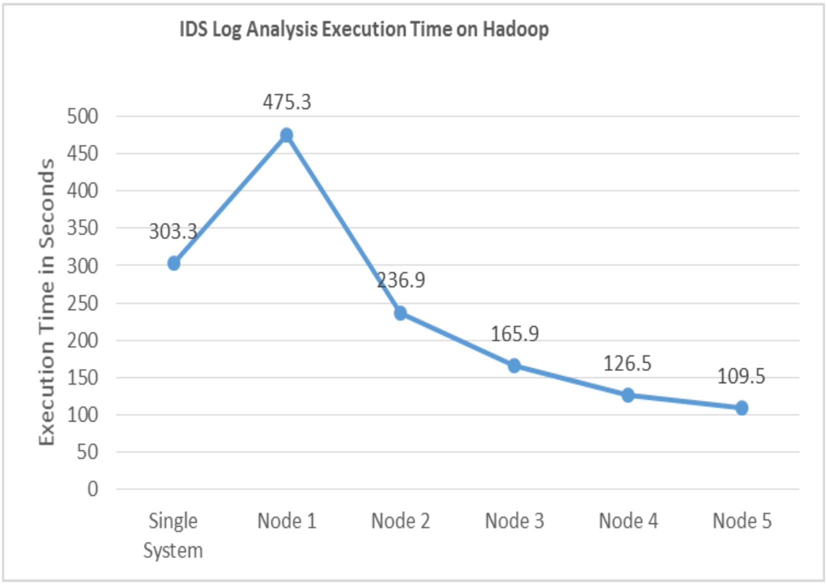

Fig 4:- IDS Log Analysis Execution Time on Hadoop

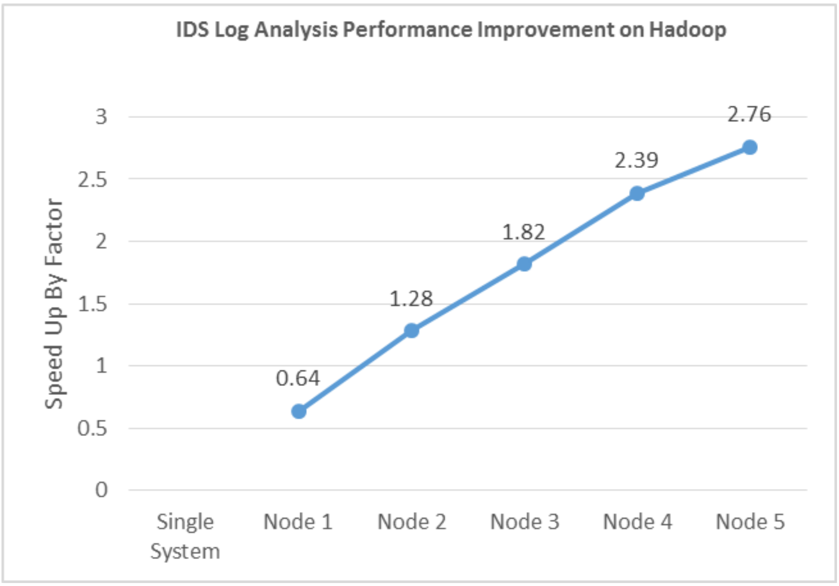

Fig 5:- IDS Log Analysis Performance Improvement on Hadoop

VI. CONCLUSION

In this paper we have explained the architectural design and performance analysis of an IDS log analysis system based on Cloud Computing infrastructure. Actually, the significant benefits to build IDS analysis system on Cloud platform are its scalability and reliability. The experimental result was positive and we found that this work can be continued with several other improvement and performance analysis.

ACKNOWLEDGMENT

I would like to thank MSRIT Management, my colleagues and Dept. of Computer Science and Applications, Bangalore University, for their valuable suggestion, constant support and encouragement.

REFERENCES

[1] Axelsson, Stefan, “Intrusion Detection Systems: A Taxonomy and Survey”, Technical Report No 99-15, Dept. of Computer Engineering, Chalmers University of Technology, Sweden, March 2000.

[2] H.Debar, M. Dacier, and A. Wespi, “Towards a taxonomy of Intrusion Detection Systems”, The International Journal of Computer and Telecommunications Networking - Special issue on computer network security, Volume 31 Issue 9, Pages 805 – 822, April 23, 1999,

[3] Holtz, Marcelo D. ; Bernardo David ; Sousa Jr., R. T. . Building Scalable Distributed Intrusion Detection Systems Based on the MapReduce Framework. Telecomunicacoes (Santa Rita do Sapucai), v. 13, p. 22-31, 2011.

[4] http://hadoop.apache.org/ (Accessed: 8 October 2013).

[5] J. Dean and S. Ghemawat, “MapReduce: Simplified Data Processing on Large Cluster”, OSDI'04 Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation - Volume 6, Pages 10-10 ,USENIX,2004.

[6] Jeong Jin Cheon and Tae-Young Choe, “Distributed Processing of Snort Alert Log using Hadoop”, IJET,Vol 5, No-3, Page 2685-2690, Jun-Jul 2013,

[7] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler, “The Hadoop Distributed File System,” IEEE 26th Symposium on Mass Storage Systems and Technologies (MSST), pp.1-10, 2010.

[8] S. Ghemawat, H. Gobioff, and S. Leung, “The Google file system, ACM SIGOPS Operating Systems Review - SOSP '03, Pages 29-43, Volume 37 Issue 5, December 2003.

[9] Yeonhee Lee and Youngseok Lee, “Toward scalable internet traffic measurement and analysis with Hadoop”. ACM SIGCOMM Comput. Commun. Rev. 43 Issue 1, Pages 5 – 13, January 2013.

[10] Martin Rehak, Eugen Staab, Volker Fusenig, Michal Pechoucek, Martin Grill, Jan Stiborek, Karel Bartos, and Thomas Engel, “Runtime Monitoring and Dynamic Reconfiguration for Intrusion Detection Systems”, Proceedings of 12th International Symposium, RAID 2009, Saint-Malo, France, September 23-25, pp 61-80, 2009.