Journal of Theoretical and Applied Information Technology
15th December 2017. Vol.95. No 23
© 2005 – ongoing JATIT & LLS

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

TOWARD SECURE COMPUTATIONS IN DISTRIBUTED PROGRAMMING FRAMEWORKS: FINDING ROGUE NODES THROUGH HADOOP LOGS

[1]N. MADHUSUDHANA REDDY,[2]Dr. C. NAGARAJU, [3]Dr. A.ANANDA RAO

[1]Assoc.Prof, RGM College of Engineering and Technology, Nandyal, AP, India.

[2]Assoc.Prof, YSR Engg.College of Yogi Vemana University, Proddatur, AP, India.

[3]Professor of CSE and Director of Academic and Planning, JNT University Ananatapur, AP, India.

E-mail: [1]madhusudhan.nooka@gmail.com, [2]cnrcse@yahoo.com, [3]akepogu@gmail.com

ABSTRACT

MapReduce programming paradigm is used to process big data in large number of commodity computers where parallel processing is leveraged. In this kind of programming phenomenon, users do not have control over distributed computations. Therefore it is quite natural for users having privacy and security concerns. The nodes involved in MapReduce computations may become malicious and cause security issues. Different attacks are possible on distributed environment. The nodes that cause such attacks are known as rogue nodes. In this paper a methodology is proposed based on analysis of Hadoop log files to find rogue nodes that are malicious and launch attacks for disturbing normal functioning of MapReduce framework. In other words, the methodology aims at integrity verification of computations in MapReduce environment. This is achieved without having any cryptographic primitives or other computational operations. Hadoop logs and low-level system calls are utilized and correlated in order to obtain operations performed by different nodes. This knowhow is then matched with system calls and invariants in program to find out malicious operations and the rogue nodes that cause such operations. The proposed methodology is evaluated with real Hadoop cluster environment to demonstrate proof of the concept. The empirical results revealed the significance of using this approach towards secure computations in distributed programming frameworks while processing big data.

Keywords: *Mapreduce, Hadoop, Detection Of Rogue Worker Nodes, Hadoop Logs, System Calls*

1. INTRODUCTION

Ever since the invention of MapReduce programming model [1] which is the paradigm shift in the way programs are executed, it is adapted by big companies like Amazon, Yahoo, Facebook and Google to mention few. It is the programming model for processing massive amount of data by supporting parallel processing executed by thousands of commodity computers and distributed file system (DFS) for supporting storage and retrieval of data. Hadoop [2] is one of the open source implementations of MapReduce programming. In spite of scalable and flexible computing services offered by MapReduce, it is known for vulnerabilities that cause attacks from adversaries. In the wake of increased cyber-attacks recent observations made by NCDRC assume significance. National Cyber Defence Research Centre (NCDRC) of India conducted National Cyber Safety and Security Standards (NCSSS) summit 2007 at Bits Pilani, Hyderabad in which the summit considered cyber-attacks as one of the cyber-crimes intentionally targeted at national assets or critical digital infrastructure [3].

As users do not have much control over outsourced MapReduce applications, it is essential to have mechanisms to secure computations in distributed programming frameworks like Hadoop. Moreover, many deceptive or malicious attempts may go undetected as the users do not have much knowledge on such activities. A survey on secure computations in distributed programming frameworks is found in [4]. Many security issues with HadoopMapReduce and possible solutions are observed in [5]. In the presence of untrusted mappers or reducers, differential privacy is used to protect privacy of big data [6]. Many researchers contributed to have secure computations. However, their study focused more on activities that need change in Hadoop framework or MapReduce operations. The effort of this paper is to devise an alternative methodology that is purely based on logs generated from time to time and system calls without indulging into the modifications of the framework.

In this paper, the main focus is on the finding of malicious activities of adversaries with deceptive and malicious behaviours by utilizing Hadoop logs and system call logs and correlating them. Without changing Hadoop framework, this paper aims to have investigations against different kinds of attacks and deceptive behaviours. Generic flow of MapReduce applications is studied with Hadoop and used it as baseline for detection of malicious or deceptive behaviour. Different invariants are used pertaining to MapReduce program execution for doing the same. Log files associated with NameNode, DataNode, JobTracker and TaskTracker are used to correlate with system call logs to find anomalous behaviour exhibited by rogue nodes. Deceptive behaviour is differentiated from malicious behaviour and explored three kinds of malicious attacks. It is obsesrved with an empirical study that execution traces of a multi-node Hadoop system can be used to find malicious worker nodes in the distributed environment. WordCount application is used as case study with a huge collection of e-Books considering as big data given input to the MapReduce application. The three attacks and difference between the deceptive and malicious nodes are demonstrated for proof of the concept.

In the literature it is found that many researchers [22], [23], [27], [28], [29] and [30] explored MapReduce programming and security of in one way or other. Out of them [22] focused on log analysis for identifying threats to MapReduce paradigm. The problem with existing works is that there needs to be a comprehensive approach that makes use of system calls and Hadoop configuration files besides logs in methodology that is missing. This paper throws light on this issue. Our contributions in this paper are as follows.

1. Review of literature is made on secure computations in MapReduce programming paradigm. These insights helped to work on rough nodes problem.
2. A methodology is proposed to identify rogue nodes that involve in MapReduce computations.
3. A threat model is used to detect different kinds of attacks on mapper and reducer in Hadoop distributed programming framework.
4. A prototype application is built to demonstrate proof of the concept.

The remainder of the paper is structured as follows. Section II provides review of literature on MapReduce programming and its security issues and solutions. Section III presents the problem formulation which is basis for the work of this paper. Section IV presents the proposed methodology which is alternative to other methods that alter Hadoop framework for secure computations. Section V presents experiments and results. Section VI provides conclusions and directions for future work.

2. RELATED WORKS

This section reviews literature on the secure computations in distributed programming paradigms. Dean and Ghemawat [7] introduced the environment in which MapReduce programming paradigm works. It provides good understanding of MapReduceenvironment where security concerns can be studied. Blanton *et al.* [8] proposed security mechanisms for outsourced sequence computations. Towards this end they used algorithms like distance computation and oblivious edit path computation. Vavilapalli*et al.* [9] explored Hadoop's compute platform known as YARN and its security mechanisms. They proposed a framework that takes care of secure computations in the context of resource navigation. Huang *et al.* [10] provided an architecture that focuses on detection of cheating nodes in MapReduce environment. Especially they provided result verification schemes in order to find malicious nodes. Zhao and Lo [11] also focused on result verification along with trust-based scheduling for secure computations. Khadke*et al.* [12] on the other hand studied system calls in cloud computing environment for diagnosing security problems through debugging. Parno and Gentry [13] proposed a system known as Pinocchio which brings about public verification scheme in distributed programming with near practical verifiable computations. Similar kind of work is done in [14].

Rabkin and Katz [15] studied Hadoop clusters and their misconfigurations. They identified and explored anti-patterns that can be used to prevent security issues in distributed computations. Braun [16] investigated on a state based approach for verification of computations in MapReduce programming. They proposed a system known as Pantry to have proof-based verifiable computations. Similar kind of work was done in [17]. Steward *et al.* [18] studied on mining corpora in distributed environments for knowledge extraction. Chen *et al.* [19] focused on a software verification primitive named as oblivious hashing for secure computations. Wang and Wei [20] proposed a framework for secure computations. It is named as Verification-based Integrity Assurance Framework (VIAF). It is meant for detecting collusive and non-collusive mappers in the Hadoop ecosystem. Dunlap *et al.* [21] built a framework known as ReVirt for intrusion detection in distributed programming models through VM logging and replay. By replaying the system before and after computations, it can detect intrusions.

Fu *et al.* [22] focused on anomaly detection in MapReduce programming. They proposed a technique known as unstructured log analysis which is based on Finite State Automaton (FSA) for anomaly detection. This work is somewhat similar to the work of this paper where explicit log analysis is made. Lou *et al.* [23] also used unstructured log analysis in distributed environment for finding mining dependency. On the other hand, Xu*et al.* [24] threw light on analysing console logs for detection of problems. Gu*et al.* [25] studied the concept of remote attestation for secure computations. Their focus was to attest the correctness of program execution remotely. Papanikolaou [26] made a review on algorithms and theory of computations. Rabin *et al.* [27] contributed towards making a scheme that verifies correctness of applications and computations. They used a model known as Evaluator-Prover. Schwarz [28] opined that in Linux distributed environments model checking can be used to detect security violations.

Xiao and Xiao [29] thought differently on security aspects. They proposed a framework known as Accountable MapReduce for that forces the machines involved in computations to help responsibility for any malicious activities. Wei *et al.* [30] proposed a framework named SecureMR which provides service assurance integrity services to prevent Denial of Service (DoS) and other attacks. Yoon and Squicciarini [31] performed log analysis for finding compromised MapReduce worker nodes. It ensures integrity and correctness of computations. Krka*et al.* [32] built a model for behavioural inference based on program invariants and dynamic execution traces using FSA. Roy *et al.* [33] proposed a security framework known as Airavat for securing MapReduce computations. It is meant for protecting system from untrusted programs. Tan *et al.* [34] built a framework known as Kahuna for diagnosing issues in MapReduce environment. It identified performance issues and security problems. Sonnek*et al.* [35] focused on finding worker nodes that are not reliable by using an adaptive reputation-based scheduling.

The review of literature revealed many insights. Some approaches in the literature focused on modifying original MapReduce functions. There are other approaches that used log analysis. However, a comprehensive methodology that has an integrated Hadoop log analysis and analysis of system calls besides using Hadoop configuration information without the need for modifying original Hadoop functionality is missing. This is the motivation behind this work which analyzesHadoop configuration file, system calls and Hadoop log files for detecting rogue MapReduce nodes.

3. PROBLEM DEFINITION

Before formulating the problem, let us examine the MapReduce framework and its modus operandi. MapReduce is a novel programming model that exploits the massive power produced by thousands of commodity computers associated with data centres in cloud computing. MapReduce framework is used to process big data. This programming model needs support of a Distributed File System (DFS) for storage and retrieval of big data. The framework has two important functions known a Map and Reduce for which developers need to write logic. Map function takes key/value pairs as input and generates intermediate output in the form of key/value pairs. The intermediate output of all mappers is given to reduce functions that run in many nodes where final output is generated. In a cloud computing environment, a master node initiates MapReduce programming. Master nodes perform two functions known as task scheduling and job management. This model is best used with Hadoop which is one of the open

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

source implementations of MapReduce. The MapReduce functionality with Hadoop is as shown in Figure 1.

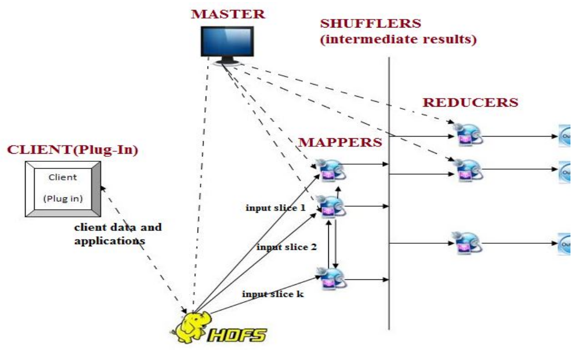

Figure 1: Functionality Of Mapreduce Paradigm

Hadoop employs master/slave model in MapReduce execution. The master node runs JobTracker and NameNode. NameNode is responsible to coordinate storage with Hadoop Distributed File System (HDFS) while JobTracker takes care of parallel processing of data with the new programming paradigm. The slave node in the framework runs DataNode and TaskTracker. The TaskTracker works in tandem with JobTracker while the DataNode works in coordination with the NameNode of master. TaskTracker coordinates multiple JVMs to track multiple map and reduce tasks that run in parallel. HDFS is the distributed file system that comes with Hadoop. As name implies it is a scalable file system in distributed environment. HDFS client is used by applications to gain access to HDFS. This file system is aware of storage racks and can help in disseminating information faster. With this scheduling becomes easier and it optimizes bandwidth usage. Both DataNode and NameNode are associated with HDFS. NameNode contains directory structure pertaining to file system while the DataNode is the node where data is stored. In a cluster data is stored in multiple locations in DataNodes. NameNode is crucial for the functioning of HDFS properly as it really reflects single point of failure for HDFS.

In this context, the problem is formulated. The worker nodes in the distributed environment may become malicious or compromised. Detecting such rogue nodes is one of the challenging problems to be addressed in this paper. Malicious nodes are to be detected by monitoring the execution dynamics at runtime. Malicious nodes might follow different execution patterns that are not with the genuine worker nodes. The aim of this paper is to detect rogue nodes by analyzing system calls and log files and examining malicious behaviour. In order to achieve certain assumptions are made. MapReduce framework is assumed correct. Worker nodes have similar hardware resources. HDFS and master nodes are trusted. Most of the workers exhibit genuine functionalities. The map code executed by different nodes produces similar output for same input. The ensuing section provides the proposed methodology to detect rogue nodes.

4. PROPOSED MTHODOLOGY

A MapReduce application is considered and generated logs when the application is running with Map and Reduce phases. The logs obtained through slave nodes are used to find peculiarbehavioural patterns that are not similar to common execution patterns. System calls are also captured by using black box testing and dynamic instrumentation without the need for changing Hadoop framework. Then the system call logs are correlated with the logs produced by Hadoop framework provides a flow of data and execution comprehensively. Dynamic instrumentation helps in extracting system traces. These traces are used to find execution flow with all minute details. This can lead to finding patterns that reflect malicious behaviour in execution. The malicious behaviour is the behaviour that violates the general operations performed by worker nodes for given MapReduce task. Another malicious behaviour is to change execution flow of Hadoop. Generally a MapReduce job contains many tasks of Map and Reduce. All tasks are related to same application but running in different commodity machines. All Map tasks should have same behaviour. In the same fashion, all Reduce tasks should have same behaviour. The expected flow of MapReduce job is as given here. Map tasks are assigned with the data obtained from DataNodes. The outcome of Map tasks is shuffled and handed over to Reduce task. Then the reducers send final output to HDFS. Each slave node participated in the distributed computing process involves in execution of a subset of tasks of MapReduce application. Thus there is some sort of temporal ordering exhibited by slave nodes.

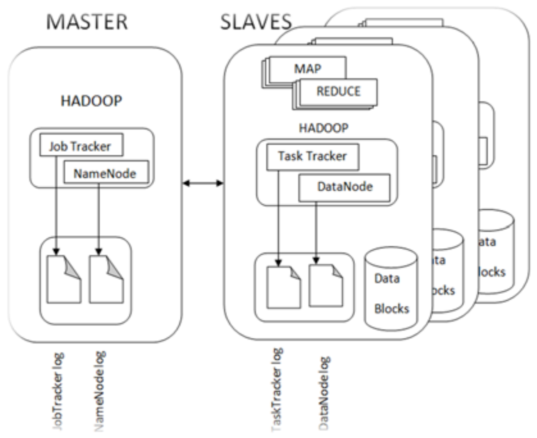

Figure 2: Master And Slaves And Different Logs Associated With Hadoop Framework

An important observation is that slave nodes get similar kind of workload with respect to a single MapReduce application and that can be easily mapped to with corresponding traces consistently. The examination of the traces with awareness of MapReduce configuration, it is possible to have good amount of knowledge on input data and setup of the system. The traces found can be compared with expected flow of execution can be correlated to reveal malicious activities that violate confidentiality, computations, availability of data of client and integrity of the whole MapReduce process. With dynamic instrumentation to get aggregated traces, system calls and logs provided by Hadoop, it is possible to detect many suspicious activities. Therefore it is intended to examine different malicious activities related to program integrity, input operations, output operations, and I/O operations.

Collection and Examination of Execution Traces of given MapReduce Application

When an application is being executed, traces of TaskTracker are obtained from slave machines and such traces are stored log files maintained by Hadoop. Thorough analysis of the log files containing interactions between HDFS and tasks related to MapReduce can reveal useful insights. Apache Hadoop makes use of Log4j for generating logs with the help of its daemon threads running in the background of DataNode, NameNode, TaskTracker and JobTracker. The logs are related to diagnostics, standard output, standard error, flow of events and other statistics. As TaskTracker is responsible to execute a subset of tasks of MapReduce application, it can provide sampling that resembles global distribution of tasks involved in the execution.

The execution patterns of Map and Reduce are subjected to thorough analysis with the help of JVM-generated logs containing system calls. Dynamic instrumentation using tools like Strace/DTrace is performed to obtain system calls from the daemon threads of TaskTracker. System calls found in such logs provide useful information related to execution flow of MapReduce tasks from JVM processes associated with TaskTracker. The log analysis is based on the important information found in the Hadoop logs. Hadoop logs contain unique identifiers for DataNode, NameNode, TaskTracker and JobTracker. Each job carried out is given a unique job id. Job ID is made up of job number and ID of JobTracker. DataNode log contains unique block id for each block of data obtained from HDFS. Every task is given unique task id which represents either Map or Reduce task being carried out. Each task ID is made up of job ID and followed by m for Map task, r for Reduce task along with the number of attempts made. More fine grained execution traces are provided by logs containing system calls. These logs are process oriented and therefore, process ID is associated with the log contents. Each JVM process associated with TaskTracker is identified by unique id known as process ID. Since it is a distributed environment, the logs have overflow or inter-leaving. Therefore timestamp field is used to arrange them in correct order of execution. The notion of drift time is used to have inspection of system events in a given time frame and synchronize the timestamps. The order of task invocations played an important role while matching Hadoop log information and system calls. For information, the flow of generic MapReduce is provided in Listing 1.

1. Worker nodes obtain configuration data from Hadoop configuration files
2. Worker nodes obtain dependent JAR files like hadoop_core-xxx.jar, logging jar file and other Java libraries.
3. Workers use taskjvm.sh for launching new JVM with all libraries needed.
4. Mappers take Map class (.class file) provided by application developer.
5. Reducers take Reduce class (.class file) provided by application developer.
6. Mappers execute Map code and produce intermediate results.
7. Reducers execute Reduce code by

```
    taking Mappers' output as input
8.  Reducers generate final result to HDFS
```

Listing 1: Execution flow of genetic MapReduce

Parsing of Log Information

The logs of Hadoop collected in distributed environment are overlapped with repetitions and it is not easy to interpret directly. Therefore the logs are parsed to have more meaningful information and get rid of repetitions. Unique identifiers and certain conventions are used in order to have more meaningful logs and system calls. MapId, Reduce ID, and Job ID are taken from log entries related to taskjvm.sh of TaskTracker. They are matched with corresponding system calls associated with a PID. The parameters of system calls are also examined in order to find any discrepancies found in dependencies and I/O operations.

Detection of Anomalies

More details on the detection of anomalies are provided here. Patterns of flow are examined and anomalies are identified. First of all, communication among HDFS client, NameNode and DataNode is considered. The HDFS client that is Map or Reduce task makes RPC call to gain access to services of HDFS. Then client establishes connection to NameNode in order to submit data to HDFS. After establishing connection, the data blocks are sent to DataNodes. Analysis of the flow is based on the communication between HDFS client, NameNode and DataNode by analyzing logs of Hadoop and logs of system calls. With the extracted network information containing TCP/IP sockets, port numbers, and IP addressed are correlated with the system calls and Hadoop logs to detect any unauthorized connection. The observation of unsuccessful socket connection related system calls, wrong IPs of ports and DataNodes indicate the sure presence of malicious behaviour in the workflow. The discrepancies between correlated information and log information also indicate malicious activity.

HDFS is used to have large scale storage and retrieval of data in distributed environment. It is especially used for data intensive MapReduce applications. The information collected from Hadoop logs is used to obtain the details of client access patterns with respect to HDFS. It includes data blocks and their location. This information is used to validate the integrity of input data. The data block ID and location of DataNode are obtained from corresponding logs. Blocks in DataNode and metadata are used by NameNode to maintain a list of blocks related to a file and list of files as well. NameNode maintains data in the form of blocks in the local file system and it also maintains corresponding metadata. The main focus is on finding location of DataNode in the form of host name or IP and also the block ID of accessed data by client. The operations on HDFS are also obtained for validating inputs given to Map task. This matching will help whether the input data is loaded from a genuine DataNode or an attacker. In the same fashion, block ID is also used for validating integrity. Listing 2 has sample NameNode and DataNode logs.

***HadoopNameNode log:**

```
STATE* Network topology has 1 racks and 2 datanodes

BLOCK* registerDatanode: node registration from ($DataNode):50010

Storage DS-624241665-192.168.1.14-50010-1382597957423

BLOCK* allocationBlock: $path blk_5905677021831100640_2283

BLOCK* addStoredBook: blockMap updated: 127.0.0.1:50010

Is add to blk_5905677021831100640_2283 size 33890
```

***HadoopDataNode log:**

```
DatanodeRegistration (DataNode ): 50010

storageID DS=624241665-192.168.1.14-50010-1382597957423,infoPort=50075,ipcPort=50020) In

DataNode.run,data=FSDatasetdirpath='($HDFS-Path)/dfs/data/current'

Receiving blk_5905677021831100640_2283 src($HDFS-Clint_IP):55950 dest($DataNode):50010

src : ($DataNode):50010, dest: (($HDFS-Clint_IP):56002, byte: 34158, op:HDFS_READ,cliID:

blockid: blk_5905677021831100640_2283 duration: 671000
```

Listing 2: Logs pertaining to NameNode and DataNode

Client interaction with HDFS shows certain patterns. These patterns are used to detect illegal access to HDFS. It may be in the form of having unauthorized access to data blocks or writing some arbitrary data blocks to DataNode and so on. Particularly the focus was on activities such as DataNodes containing data file in the form of blocks provided by client and loading the same into Map task that come from a DataNode holding duplicated data blocks. The traces of system calls pertaining to DataNode are analyzedto detect activities between DataNode and HDFS client that resulted in data transfer. The presence functions like open(), accept() provide the presence of system calls to access data and establish connection respectively. As HDFS communicates in the form of TCP/IP sockets the system calls can be understood with ease. For instance sendfile() system calls having file descriptors as arguments helped to check whether HDFS client get correct data block from DataNode. System call traces on the DataNode are as shown in Listing 3.

```
accept() = out_fd;

open("pathname(blockID)", O_RDONLY) =
in_fd; sendfile(out_fd,in_fd,offset,bytes) =
bytes;
```

Listing 3: System call traces associated with DataNode

The dynamic execution traces of MapReduce application are shown in Listing 2. These trace are collected by using DTracing tool that extracted traces of TaskTracker's associated JVM processes at system calls level and method level. This is done in each slave node. The program execution flows are also used along with corresponding semantics from the traces containing information such as the classes that have been loaded into Map or Reduce task, and the operations performed over there. The data flow and control flow of MapReduce are thus understood at lower granularity. This information can be analysed further semantically for finding causal relationships between components associated with MapReduce application. Thus the causal relationship provides useful insights on the execution logic provided in the Map and Reduce classes of the application considered.

The traces and correlation procedures aforementioned are made effective further by correlating them with certain trust-worthy patterns of MapReduce workers. It is related to program invariants that are associated with different workloads. The invariants help to analyse log sequences to know whether the execution process differed with anticipated workflow. When the log files are not complying with invariants anomaly can be suspected. The invariants considered in Hadoop environment include presence of libraries pertaining to Java and Hadoop, fetching of intermediate results by Reduce nodes, writing intermediate results by Reduce nodes, and reading configuration files consistently. As libraries of Hadoop are independent of MapReduce applications, the activities pertaining to loading libraries in worker nodes should be consistent. The log analysis also provides ample insights related to the loading of all necessary libraries to execute Map and Reduce tasks. HDFS client access patterns with HDFS can also provide information to check against known invariants. The configuration file holds information related to storage location while the parameters in system call provide where actually the Mapper wrote the generated intermediate output. When these two locations are not matching, it is an indication of malicious behaviour. The events provided in Listing 4 can show the interactions between HDFS and MapReduce.

- HDFS client sends data file to HDFS where data is split and stored in DataNodes in the form of blocks
- Mapper takes data blocks from DataNodes
- Reducer sends the final results to HDFS

Listing 4: Order of events reflecting interaction between HDFS client and MapReduce

The events are performed in the given order. The order reflects workflow and that flow are never changed. In addition to this different invariants of MapReduce are used to have more comprehensive understanding of behaviour of different functions. Unless Map function is under the influence of an attacker, it produces same output with same input. The method invocations should be same when there is same execution flow. It is also possible to consider many invariants at a time. One such example is that a Map function writing its output to local disk/HDFS is always preceded by the same Map function loading data blocks from HDFS. In the same fashion Reduce function writes final output to HDFS of local disk only after taking intermediate output produced by Map function. This kind of execution order can help in detecting rogue nodes and malicious behaviour. Another important analysis is to check the number of system calls and number of return system calls should be same for both Map and Reduce tasks. Finding the different in the number is the indication of anomaly or an attack named as file descriptor attack.

Inputs and Computation Integrity Checking

As mentioned earlier the integrity checking process is based on the correlation between logs of Hadoop and logs containing system calls. Invariants discussed in the previous section are used to continue with integrity checking. By using input data being used by worker nodes it is possible to check input integrity. JabTracker log provides very important information for achieving this. It includes data size of the input and the number of parts into which it is split. The TaskTracker log provides information like file name, map task, and path in HDFS. It is possible to obtain MapID, JobID and TaskID besides the location of nodes and data being used by Map tasks. By correlating the log information of JobTracker and TaskTracker and log events of I/O can provide relationships between MapReduce tasks and HDFS. Simple act of checking input data size provided in the JobTracker against that of HDFS. Moreover, the block ID and location of a block and the usage patterns of Map task and I/O events can provide useful insights. Correlation of logs can also be used to check the similarity of outputs produced by mappers and reducers. HDFS access patterns and the MapReduce task operations are correlated and anomalies are detected.

With respect to checking integrity of computations, it is possible to have compromised nodes or rogue nodes as mappers. Though they get valid input from the HDFS, they may run malicious code for producing incompatible computational output. In order to check such integrity, execution behaviour of mapper is analyzed with the help of TaskTracker logs and corresponding system call logs or worker nodes. Though Hadoop logs provide information about MapReduce dynamics, they cannot provide system calls information. When system calls are used for correlation, the computational integrity can be verified. Sequence of system calls across worker nodes are verified to find out discrepancies due to the presence of malicious worker nodes or rogue nodes. For instance a malicious node is the one which tries to conserve resources or for any gain by using less number of write() when compared with an honest node. An important observation is that system call statistics are not sufficient the analysis needs to be coupled with program semantics and execution flow as malicious nodes often do not follow normal execution flow.

5. EXPERIMENTAL SETUP AND RESULTS

Hadoop 2.7.x is use with Ubuntu operating system installed in virtual box. It is configured as cluster. The system traces of worker nodes are aggregated and logs are obtained. The Hadoop logs are related to JobTracker and TaskTracker daemons. By taking a simple word count application, the Map and Reduce tasks are executed and traces are collected from log files. The log files are also collected from NameNode and DataNode. Using Dtrace, captured system calls that are related to interactions between MapReduce and HDFS containing I/O operations are captured. As said before, WordCount application is used for empirical study. This application is well known to developer community who use distributed programming frameworks like Hadoop. The application counts the occurrences of different words in a given input file or set of files. The experiments are made and correlation of logs is done as explored in details in the methodology. Figure 4 illustrates inputs, execution process and final output of WordCount application. In the experiments made it is observed patterns that reflect strange or malicious behaviour.

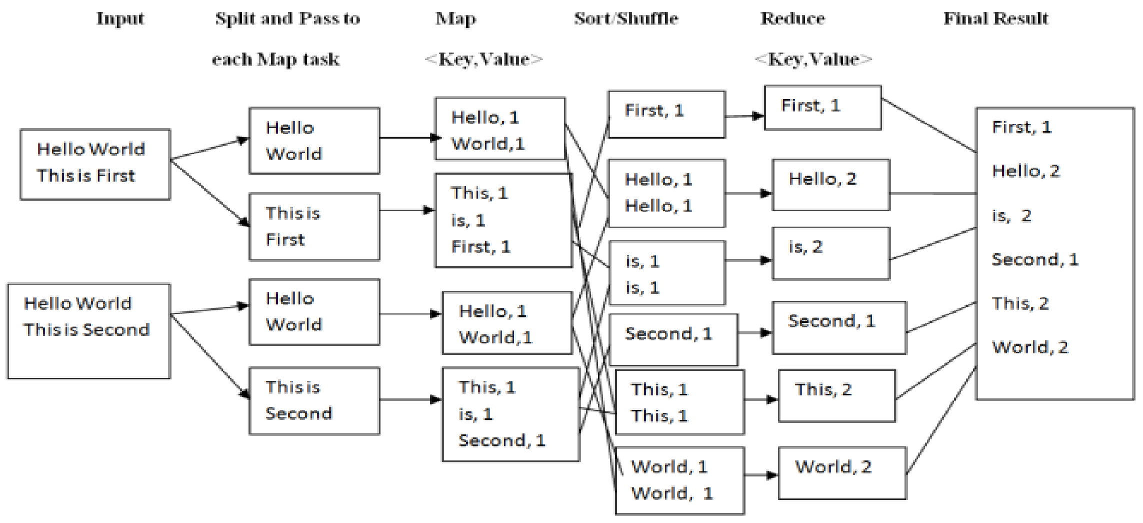

Figure 3: Illustrates Mapreduce Tasks Of Wordcount Application With Inputs And Outputs

As shown in Figure 3, it is evident that the inputs, intermediate results of Map, Shuffle and Reduce are presented with respect to WordCountMapReduce application. Two kinds of workers that are not honest are considered. They are known as deceptive worker and malicious worker. Deceptive node is the node which skips some computational flow in order to save computing resources. On the other hand a malicious worker node is the node that attack MapReduce workflow in different ways. Suspicious JVM launching is one of the attacks. As the JVM launching is made by TaskTraker, the TaskTracker log file and the system calls related to JVM launching are used to detect this kind of attack. The JVM ID, Map ID, JobID of Hadoop logs and the call log information related to nodes should have similar values. Violating this is the indicative of such attack.

Another attack is related to injection of malicious JAR files. As the Hadoop makes use of known JAR files and classes consistently, verification of malicious JAR is done by examining classes loaded for Map and Reduce tasks. By obtaining location and name of Java class files from logs, the malicious worker node's behaviour is detected. Yet another malicious attack is the misplaced intermediate outputs that are generated by Map tasks. This attack is detected simply finding the location of intermediate output by examining log files. When the mapper is writing intermediate output to a location other than the location to which it needs to write as per configuration, it is detected as rogue node or malicious node. Attacks can also be made by changing Hadoop configurations. When the MapReduce activity is underway, an attacker may modify configuration files like hddfs.site.xml, mapred-site.xml, and core-site.xml. This is effectively detected by comparing system calls and Hadoop log contents.

Experimental Results

This section provides results of experiments with respect to rogue nodes that have deceptive and malicious behaviours. The results are based on the case study taken with WordCount application. Six nodes are used in the experiments where mappers are executed. One of them is made deceptive or malicious node. E-books collected from free source named Qutenberg [36] are used as dataset or input to MapReduce application that WordCount. The application is responsible to take big data (bulk of e-Books) and produce word count output. In the process it makes use of parallel processing power of distributed programming framework Hadoop configured with 6 worker nodes. Two scenarios are used for experiments. They are deceptive behaviour scenario and malicious behaviour scenario. In the case of the former, the size of output produced by Map task of deceptive node found in the TaskTracker log file is significantly different from that of genuine nodes. The size shown in log file is 10 bytes while the actual size of output produced by genuine mapper is 304320 bytes. This difference in figures reflects only deceptive behaviour of worker node. However, it does provide execution behaviour of Map task. Additional insights are obtained by analyzing system calls and comparing with log information related to the system calls. The log content of system calls show much difference in invocation of write() call in the Map task. The count of calls to write() of deceptive node is much lesser than that of honest worker node. However, the calls related to read() are found similar. The rationale behind this is that the deceptive mapper does not execute loop or skip it to avoid computations. Many experiments are made and the results are as follows. System calls of related write() of genuine mappers is 722 out of the total number of read/write calls 43580. Number of write() calls of deceptive worker node is 190 out of 44520 read/write system calls.

In case of malicious behaviour scenario, it is not sufficient to have statistics. Therefore Hadoop and system calls are to be correlated by considering the generic workflow of MapReduce provided in Listing 1. According to the generic workflow three events are identified to be malicious. First, the generic workflow is changed by attacker by launching his own JVM for supporting malicious activities. Second, it is observed that changing map code (logic of counting) can influence the intermediate output of Mappers in WordCount application. It is tested by changing Map class code. The third one studied is the case in which input data is taken from a location which is not authentic. In case of the first malicious behaviour, when adversary tried to have his own JVM, the logs of TaskTracker provides JVM ID, Map ID, and Job ID are studied to know the behaviour of malicious worker node compared with honest node. The JVM behaviour of Hadoop and that of attacker differ which is the indication of malicious attack.

With respect to the second malicious behaviour, it is observed from the log files that the malicious nodes showed 20% of write() calls while the other nodes show 68-80% write() system calls. This is the indicative of malicious behaviour. In the third malicious behaviour case, the code in Mapper class is altered. Instead of reading data from HDFS configured location, it reads from local directory. The logs contain descriptor information of files related to read() system call to detect this kind of malicious behaviour. The difference in file path and file prove the presence of such malicious attack.

6. DISCUSSION

This paper threw light into a methodology based on Hadoop logs, system calls and Hadoop configuration files. This methodology is meant for identifying rogue worker nodes that participate in MapReduce programming in distributed environments. Unstructured log analysis using FSA [22], and log analysis for mining dependencies [23], and log analysis for finding compromised MapReduce nodes in distributed environment [31]. These methods followed log analysis approaches. However, we found that these approaches can be improved further. Towards this end, this paper threw light on a comprehensive methodology that not only considers log files but also system calls made from Hadoop and Hadoop configuration files that can influence the series of functions carried out in the process of execution. The work of this paper produced accurate results when compared with the existing methods. The rationale behind this is that the use of multiple ingredients to make conclusions on the identification of rogue nodes.

7. CONCLUSIONS AND FUTURE WORK

In this paper, an alternative methodology is presented which is on the contrary to other approaches that force changes in Hadoop framework to detect malicious worker nodes. The methodology focuses on log files associated with NameNode, DataNode, JobTracker and TaskTracker in a multi-node Hadoop cluster environment. The behaviour of rogue nodes is divided into deceptive and malicious behaviours. Deceptive behaviour is understood as the behaviour of a rogue node that skips certain computations for saving computational resources. Whereas the malicious worker nodes is the compromised node that intentionally makes attacks on MapReduce programming paradigm. These attacks are considered in the attack model and experiments are made to prove the concept. A case study application known as WordCount is used with big data for processing. Dataset given as input contains thousands of e-Books collected from Qutenberg project. With this case study, the methodology is evaluated in terms of detecting deceptive and malicious behaviours of rogue worker nodes in a distributed environment. The empirical study revealed the significance of the methodology in detecting rogue nodes toward secure computations in distributed programming frameworks while processing big data. Further investigation is needed to study the situations where multiple nodes are compromised. Another research direction is to characterize rogue DataNode that may create snapshots of legitimate nodes and re-introduce altered copies. This is a straight forward attack that is difficult to detect. Continuous efforts are needed to keep MapReduce tasks to remain secure and it is an open problem to be addressed. It is left for future work.

REFRENCES:

[1] Apache Software Foundation. (2016). *MapReduce Tutorial.* Available: https://hadoop.apache.org/docs/stable/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html. Last accessed 01 December 2016.

[2] The Apache Software Foundation. (2016). *Welcome to Apache™ Hadoop.* Available: http://hadoop.apache.org/. Last accessed 01 December 2016.

[3] NCDRC (2017). National Cyber Safety and Security Standards. Available online at: https://www.ncdrc.res.in/national-cyber-safety-and-security-standards-summit-2017.php [Accessed: 20 February 2017]

[4] Madhusudhan Reddy, N., and Nagaraju, C. (2015). Survey on Emerging Technologies for Secure Computations of Big Data. I-Manager’s Journal of Cloud Computing, 2 (1), p1-6.

[5] Priya P. Sharma and Chandrakant P. Navdeti. (2014). Securing Big Data Hadoop: A Review of SecurityIssues, Threats and Solution. *International Journal of Computer Science and Information Technology*.5 (2), p1-6.

[6] Madhusudhan Reddy, N., and Nagaraju, C., AnandaRao A., (2017)“ Protecting Privacy of Big Data in presence of untrusted Mapper and Reducer”, Indian Journal of Computer science & Engineering, ISSN No: 0976-5166, Vol 8, No 3, June – July 2017, p201-209.

[7]. Jeffrey Dean and Sanjay Ghemawat. (2006). MapReduce: Simplified Data Processing on Large Clusters. COMMUNICATIONS OF THE ACM.51 (1), p107-113.

[8]. Marina Blanton ,Mikhail J. Atallah , Keith B. Frikken , and QutaibahMalluhi(2012)Secure and Efficient Outsourcing of Sequence Comparisons, p1-18.

[9]. Vinod Kumar, VavilapalliMahadev ,KonarSiddharth Seth, Arun Murthy Robert, Evans BikasSaha, Thomas Graves and Carlo Curino . (2013). Apache Hadoop YARN: Yet Another Resource Negotiator, p1-16.

[10]. Chu Huang, Sencun Zhu and Dinghao Wu (2012)Towards Trusted Services Result Verification Schemes for MapReduce., p1-8.

[11]. Shanyu Zhao and Virginia Lo. Result Verification and Trust-based Scheduling in Open Peer-to-Peer Cycle Sharing Systems. p1-10.

[12]. Nikhil Khadke, Michael P. Kasick, Soila P. Kavulya, Jiaqi Tan, PriyaNarasimhan (2012) Transparent System Call Based Performance Debugging for Cloud Computing, p1-6.

[13]. Bryan Parno ,Jon Howell ,Craig Gentry Mariana andRaykova. (2013). Pinocchio: Nearly Practical Verifiable Computation.IEEE, p1-15.

[14]. Bryan Parno,CraigGentry,Jon Howell and Mariana Raykova. (2013). Pinocchio: Nearly Practical Verifiable Computation. IEEE.p239-252.

[15]. Ariel Rabkin and Randy Katz EECS Department andUC Berkeley.(2012). How Hadoop Clusters Break, p1-12.

[16]. Benjamin Braun, Ariel J. Feldman ,ZuochengRen, SrinathSetty, Andrew J. Blumberg, and Michael Walfish. (2013) Verifying computations with state, p1-17.

[17]. Benjamin Braun, Ariel J. Feldman?,ZuochengRen, SrinathSetty, Andrew J. Blumberg and Michael Walfish. (2013). Verifying computations with state. p341-357.

[18]. Gordon Stewart,Gregory Crane and Alison Babeu. (2007). A New Generation of Textual Corpora Mining Corpora from Very Large Collections. p1-10.

[19]. Yuqun Chen,„ RamarathnamVenkatesan,„ Matthew Cary, Ruoming Pang, SaurabhSinha and Mariusz H. Jakubowski,„. (2003). Oblivious Hashing: A Stealthy Software IntegrityVerificationPrimitive. springer. p1-15.

[20]. Yongzhi Wang and Jinpeng Wei. (0). VIAF: Verification-based Integrity Assurance Framework for MapReduce. p1-8.

[21]. George W. Dunlap, Samuel T. King, SukruCinar, Murtaza A. Basrai and Peter M. Chen. (2002). ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay. Symposium on Operating Systems Design and Implementation.p1-14.

[22]. Qiang FU, Jian-Guang LOU, Yi WANG and Jiang LI. (2014). Execution Anomaly Detection in Distributed Systems through Unstructured Log Analysis. p1-11.

[23]. Jian-Guang LOU, Qiang FU, Yi WANG and Jiang LI. (0). Mining Dependency in Distributed Systems through Unstructured Logs Analysis. p1-6.

[24]. Wei Xu, Ling Huang, Armando Fox, David Patterson and Michael Jordan. Online System Problem Detection by Mining Patterns of Console Logs. p1-10.

[25]. Liang Gu ,Xuhua Ding, Robert H. Deng, Bing Xie and Hong Mei. (2008). Remote Attestation on Program Execution. p1-9.

[26]. Mikhail J. Atallah and Marina Blanton. (2010). Algorithms and Theory of Computation Handbook. Nick Papanikolaou. 2 , p1-4.

[27]. Michael O. Rabina,Rocco A. Servedio and Christopher Thorpe. Highly Efficient Secrecy-Preserving Proofs of Correctness of Computations and Applications. p1-14.

[28]. Benjamin W. Schwarz. Model Checking An Entire Linux Distribution for Security Violations. p1-39.

[29]. Zhifeng Xiao and Yang Xiao. (2011). Accountable MapReduce in Cloud Computing. IEEE.p1099-1104.

[30]. Wei Wei, Juan Du, Ting Yu and XiaohuiGu. SecureMR: A Service Integrity Assurance Framework for MapReduce. p1-10.

[31]. Eunjung Yoon and Anna Squicciarini. Toward Detecting Compromised MapReduce Workers through Log Analysis. p1-10.

[32]. Ivo Krkay, YuriyBrunx, Daniel Popescuy, Joshua Garciay and NenadMedvidovic. (2010). Using Dynamic Execution Traces and Program Invariants to Enhance Behavioral Model Inference. p1-4.

[33]. Indrajit Roy, Srinath T.V. Setty, Ann Kilzer, VitalyShmatikov and Emmett Witchel. Airavat: Security and Privacy for MapReduce. p1-51.

[34]. Jiaqi Tan, XinghaoPan,EugeneMarinelli, SoilaKavulya, Rajeev Gandhi and PriyaNarasimhan. Kahuna: Problem Diagnosis for MapReduce-Based Cloud Computing Environments. p1-8.

[35]. Jason Sonnek†, Abhishek Chandra and Jon Weissman. (2007). Adaptive Reputation-Based Scheduling on Unreliable Distributed Infrastructures.p1-30.

[36] Qutenberg. Project Qutenberg. Available online at: https://www.gutenberg.org/. [Accessed: 20 February 2017].