A Cloud-based Malware Detection Framework

https://doi.org/10.3991/ijim.v11i2.6577

Eman Ahmed*
Ain Shams University,Cairo, Egypt
e.ah.saad@gmail.com

Amin Sorrour*
Misr University for Science and Technology, 6th of October, Egypt
asorrour@yahoo.com

Mohammed Sobh
Ain Shams University,Cairo, Egypt
mohamed.sobh@eng.asu.edu.eg

Ayman Bahaa-Eldin
The British University in Egypt, Cairo, Egypt
ayman.bahaa@bue.edu.eg

Abstract—Malwares are increasing rapidly. The nature of distribution and effects of malwares attacking several applications requires a real-time response. Therefore, a high performance detection platform is required. In this paper, Hadoop is utilized to perform static binary search and detection for malwares and viruses in portable executable files deployed mainly on the cloud. The paper presents an approach used to map the portable executable files to Hadoop compatible files. The Boyer–Moore-Horspool Search algorithm is modified to benefit from the distribution of Hadoop. The performance of the proposed model is evaluated using a standard virus database and the system is found to outperform similar platforms.

Keywords—Cloud computing, Security issues, Malware, Static Binary Search, BMH, Hadoop.

1 Introduction

Scanning files for viruses in a rapid manner can be achieved by utilizing Hadoop facilities. Hadoop provides parallel working mechanism. However, Hadoop was mainly created to deal with large data-sets. Most viruses and malwares exist in Portable Executable (PE) small files or images, which can affect Hadoop performance dramatically and thereby search performance. In this paper, a system is presented to carry out PE files static search using hadoop and is organized as: *section2:* Related Work, *section3:* The System Environment (the environment used, hadoop, infected files and DB used), *section4:* System General Architecture, *section 5:* Factors Affect-

ing Performance during Testing Phase, *section6:* The system architecture details and results and *section7:* Running on Virtual Multi-node Cluster *section8:* Concluding the work done and the future work.

2 Related Work

A vast amount of small files are used across the cloud as PE files, images. Since cloud environment inherited internet properties, cloud environment is vulnerable to malwares. Hence, a demand to make researchers study and improve different techniques for scanning of files across clouds in a fast manner. Researchers as (1) handled this problem using hadoop environment too. But they proposed architecture of how hadoop framework uses its daemons to cooperate in scanning without further details. Other Researchers (2) handled the static search using Hadoop and described how to utilize pre-existing tools together with hadoop without describing how the searching is handled by antivirus programs, what algorithms could be used, and how to deal with the antivirus DB. Researchers in (3) described in a very good way how ClamAV antivirus works and presented a detailed description for the DB file. They presented a scanning technique to search statically in ClamAV DB but not in cloud environment. Hence, there is another need to understand practically these details as it might be helpful for more enhancements to be carried out concerning this field. This paper presents an approach to better understand how to utilize hadoop facilities in signatures static search, how to use an antivirus DB to perform this search, what algorithm can be chosen, what other factors could affect environment performance and how to overcome them.

3 The System Environment

3.1 The Environment Used

Clouderaquickstart vm is used. Monitoring and General Configurations for MapReduce Jobs can be done through Cloudera Manager and Hadoop Tracking Interface. Java code is used in writing the application program.

3.2 Hadoop

Hadoop MapReduce is a software framework used for creating applications dealing with vast amounts of data in-parallel on large clusters. A job divides dataset into independent chunks to be processed by the map tasks in a parallel manner. Then a sorting to the outputs of the maps is done to form input to the reduce tasks. In atypical system the input and the output of a given job are stored in a file-system. The framework schedules tasks, monitors them and re-executes any failed tasks.

Hadoop MapReduce Jobs can be implemented through many ways using scripts or coding. In this case study Java coding was used to create the Jobs with eclipse IDE**.**

To run Hadoop Jobs, three modes can be used (4) (5):

Standalone (local Mode): Hadoop uses local file system instead of HDFS and have one-mapper and one-reducer. Pseudo-Distributed Mode: All daemons run on single machine and mimic the behaviour of cluster. All daemons run locally and use HDFS. Multi-mappers and Multi-reducers. Fully-Distributed Mode: Hadoop running on real clusters. In this case study, the pseudo-distributed mode is used.

As a general term, Daemons means a process running in the background. Hadoop has five daemons: NameNode, Secondary NameNode, DataNode, JobTracker and TaskTracker.

3.3 Infected Files and DB Used

To create samples of infected executable files, a tool was developed using Java to take input clean files and inject virus signatures in random places in the files. The first signature used was that of eicar test file (6). This file has amazing benefits, one can test virus scanners using it, has no dangerous effect, and its signature exists in already known antiviruses DB sets as ClamAV, Symantec and many others. The infected executable files, created in this case study, were infected by many other viruses listed in ClamAV antivirus DB (7). These samples were tested by online scanner to check if they can be found by already known free scanners using VirusTotal Online Scanner (8).

4 System General Architecture

This case study was done to achieve a system capable of utilizing Hadoop facilities to speed up binary scans on infected files. The System General Architecture is shown in the following figure *(Fig1)*. As shown in the figure, the architecture is dealing with input files to be uploaded in the HDFS to be scanned using DB files by mappers. Then, a final report is formed by reducer for the scan results.

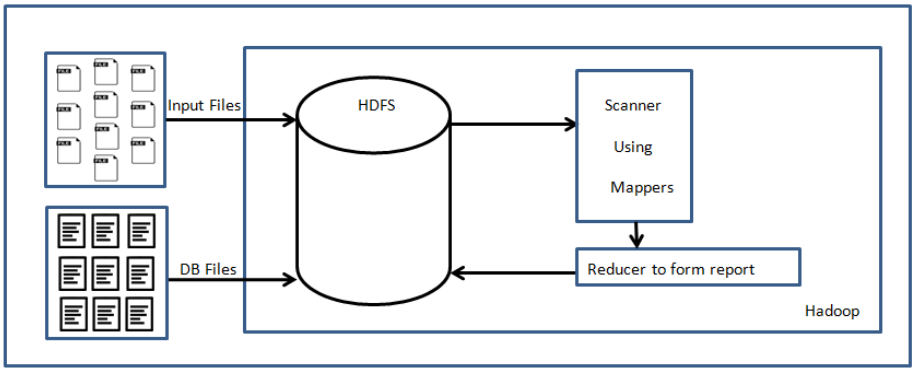


Fig. 1. System General Architecture

5 Factors Affecting Performance during Testing Phase

In the section, the factors affecting the Hadoop byte search are introduced together with sample tests done to prove them.

The factors affecting the scanning performance in Hadoop environment:

- Resources and Configurations
- Algorithm used to scan the files
- Size of Files
- DB organization and location

5.1 Resources and Configurations

The ClouderaquickstartVM was set to have 8GB Ram and 2CPU. Cloudera Manager is used to adjust some yarn configurations. Some default settings for memory allocation to mappers and reducers were adjusted as: "yarn.app.mapreduce.am.resource.mb", "mapreduce.map.memory.mb" and "mapreduce.reduce.memory.mb" 2GB. Java Heap Size 512MB. These modifications were done to speedup performance and avoid memory usage errors.

5.2 Determining a Search Algorithm

Naïve Brute Force Algorithm: Naïve algorithm *(Fig2)* (9) is a very simple algorithm, sometimes the first one that comes to mind. It is simply, checking the occurrence of a pattern inside the bytes of a file, element by element to see if a match exists. So first, it checks the first element in the pattern against the first element of the file array; if not, check it against the next element in the file array, and so forth. In the worst case, searching using naïve algorithm takes O(nm); n is the length of the file array and m is the length of the pattern.

```
Loop1 ( i from 0 to n-m)
    Loop2 (j from 0 to m)
        if (file[i + j] != pattern[j]) break;
    end  Loop2
    if (found)
       report the occurrence of the pattern at defined index
end Loop1
    // no match found if we reach this point
```

Fig. 2. Naïve Brute Force Algorithm

Test1:
Was done using 4 exe input files and Naïve Algorithm *(Fig3)*.

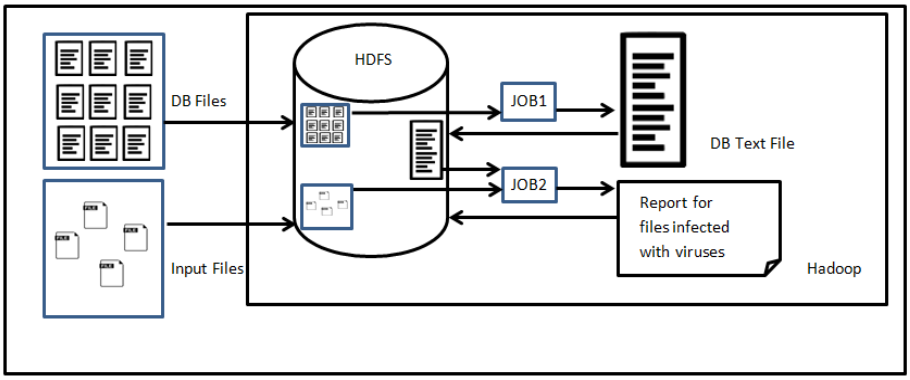


Fig. 3. Test1 Detailed Jobs' Architecture

Results of Test1:
The four files' sizes are F1 (11.5KB), F2 (811bytes), F3 (425KB) and F4 (421bytes). The searching phase took 4hrs. Hence, need for boosting the speed of search. The first thing to think about is changing the search algorithm as done in test2.
The performance of Test1 *(Table1)* on pseudo-distributed mode was checked using Cloudera Manager and Hadoop Tracking Interface.

Table 1. Test1 - Naïve Search Performance for 4 Portable Executable Files

JOB ID	JOB Description	Elapsed Time	INPUT	OUTPUT
JOB1	AV-signature preparation phase	2mins:45secs	9 DB files	1 Text DB File
JOB2	Searching PE files for viruses and reporting results phase	4hrs:44mins:27secs	4 files and 1 Text DB file (read from HDFS)	Report with scan results

Boyer–Moore-Horspool Algorithm: BMH algorithm *(Fig4)* (10) (11) is a fast search algorithm originally done to check the occurrence of a pattern in a given Text. It pre-processes the pattern to produce a jump table containing, the number of characters that can be skipped. The preprocessing in pseudocode is as in the shown figure *(Fig4)*. And the search function reports the index of the first occurrence of the needle (pattern) in haystack (file bytes).

```
Preprocessing(pattern)
        Create Table T for 256 integers.
        T[i] = length(pattern)
        for i from 0 to length(pattern) - 1
        T[pattern[i]] = length(pattern) - 1 - i

Searching (needle, haystack)
        T = Preprocessing(needle)
        skip = 0
        while length(haystack) - skip ≥ length(needle)
        i =length(needle) - 1
        while haystack[skip + i] = needle[i]
        if i = 0
        return skip
        i = i - 1
        skip = skip + T[haystack[skip + length(needle) - 1]]
        return not-found
```

Fig. 4. Boyer–Moore-Horspool Algorithm

Test2:

Was done using the same 4 exe input files and the Boyer–Moore-Horspool Algorithm *(Fig5)*.

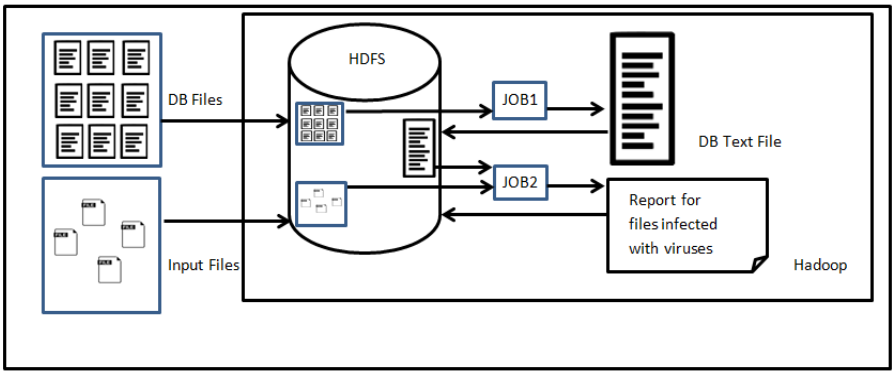


Fig. 5. Test2 Detailed Jobs' Architecture

In this test the same idea is implemented but using byte array search instead of string in Text. The search is done from right to left in the pattern. If the first element, did not find a match it uses the jump table to skip and search in another index in the file. If the first element, has a match in the file move to next element in the pattern to the left and so forth till the whole pattern matched. Previous searches (11) proved

that, this algorithm is considered to be the fastest in byte searches and that is why it is chosen to be used in this case study.

Results of Test2:

For the same four files: F1 (11.5KB), F2 (811bytes), F3 (425KB) and F4 (421bytes). A time reduction took place in the searching phase from 4hrs to 2hrs. There is a slight reduction in time but still not enough. From here, another factor is affecting the searching performance. This could be the size of files, as it will be explained next.

The performance of Test2 *(Table2)* on pseudo-distributed mode was checked using Cloudera Manager and Hadoop Tracking Interface.

Table 2. Test2 - BMH Search Performance for 4 Portable Executable Files

JOB ID	JOB Description	Elapsed Time	INPUT	OUTPUT
JOB1	AV-signature preparation phase	2mins:45secs	9 DB files	1 Text DB File
JOB2	Searching PE files for viruses and reporting results phase	2hrs:57mins:51secs	4 files and 1 Text DB file (read from HDFS)	Report with scan results

5.3 Size of Files

The Hadoop Distributed File System (HDFS) and MapReduce are mainly optimized for processing and storing large files. Small files in HDFS reduce the Hadoop general performance. A file is called small when its size is less than the HDFS block size, which is 64 MB by default. From here one can define a block size as, the smallest unit of data that a file system can store. Hence, storing a file of size 1k or 60Mb, will occupy one single block. Once the file size crosses the 64Mb boundary, a second block is needed and so on.

Map tasks usually process a block of input at a time. If the file is very small and there are a lot of them, then each map task processes very little input, and there are a lot more map tasks, each of which imposes extra overhead. For example a big file as 1GB file is broken into 16 blocks (each 64MB). However, in case using many small files as 10,000 of 100KB files, the job time can be tens or hundreds of times slower than the equivalent one with a single input file. This is because each file from the 10,000 files uses one map task. Many blocks means, lots of traffic. Where each request for a given block, recommends a processing by the Name Node to figure out where that block can be found. Unfortunately, most PE files and images are all less than the HDFS default block size. One Solution to this problem is using Hadoop Sequence file. SequenceFiles are like containers for smaller files. Packing files into a SequenceFile makes storing and processing the smaller files more efficient. This way is handled as coming next.

Test3:

In this Test a sequence file for 10exe files is used as input. The files of sizes: F1(3.2MB), F2(11.5KB), F3(17.2KB), F4(17.2KB), F5 (811bytes), F6(496.2KB), F7(496.2KB), F8(425KB), F9(381bytes) and F10(421bytes). To improve the search-

ing phase, instead of using one mapper for one sequence file with key,value pairs as <FilePath,FileBytes>. Two reducers were used, to split the input sequence file into two sequence files to have two mappers working in parallel. This searching took about 2mins to search the 10 files.

A comparison between the above three tests during the execution of the scanner JOB is summarized in the next chart *(Fig6)*. The scanner JOB in Test1, Test2 and Test3 is JOB2.

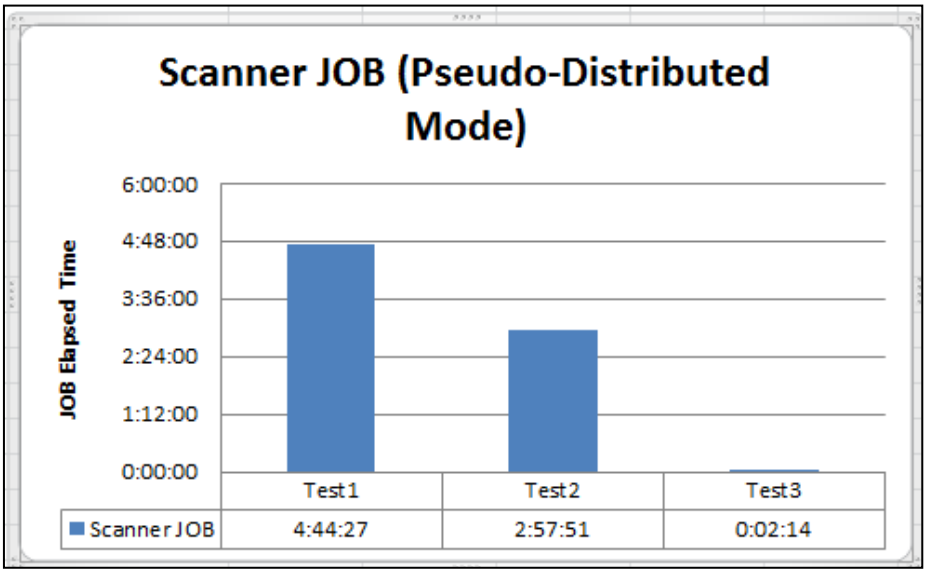


Fig. 6. Scanner JOB of the Three Tests in Pseudo-Distributed Mode

From this comparison, the application written in Test3 was the best as it has the minimum scan time in a pseudo-distributed mode. It has the minimum scanning time with a larger number of input files (10 files), compared to Test1 and Test2 where 4 input files only were used. A further explanation for the steps of Test3 and results is discussed in section 6. In section 7, Test3 application is tried again but in a virtual multi-node cluster.

5.4 DB organization and location:

During all tests the Clam-AV virus signatures database was used. They are 9 files each holding 1000 record with sizes ranging from 92.5KB to189KB. During Test1 and Test2: the DB 9-files were reduced into one Text file this job occupied 2mins *(Table1) (Table2)*. In Test3: were gathered in one sequence file for being all small in size key,value pairs as <VirusName, VirusSignature> and occupied 33secs *(Table3)*.

One Issue remained left was the place of the formed DB sequence file. Two ways can be used, either leaving it in HDFS the way it is or in Distributed cache. The advantage of using HDFS is we can store large files in it. However, a massive problem

may occur as DB should have limited resources of connections. Having the DB in HDFS means more connections and calls to the DB from DataNodes. This in return leads to slower performance and eventually DB bottleneck.

The Distributed cache has a maximum limit of 10GB (12). The framework will copy the necessary files on to the DataNodes before processing any tasks of a job on any of these DataNodes. Its efficacy comes from the fact that the DB file is copied once per job. One more advantage is that since it is RAM/memory based the files used are destructed when the job completes.

In this case study, we are dealing with DB of (1.2 MB) in total. From here, the choice of Distributed cache fits more. Where the file is copied to caches of DataNodes to make them search locally and relieve the congestion on DB. Test1,2 were done using MapReduceV1, while Test3 code was done using MapReduceV2 to be able to use distributed cache.

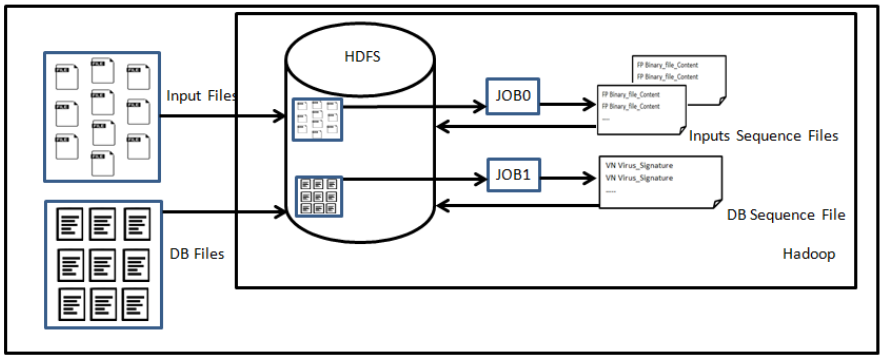
6 The System Architecture Details and Results

The System Detailed Architecture is shown in figures *(Fig7a, Fig7b)*. Three JOBs are used. JOB0, forms the input sequence file that containes the key,value pairs <FilePath,FileBytes>. JOB1, forms the db sequence file that containes the key,value pairs <VirusName, VirusSignature>. JOB2, is resposible for the searching phase and it has the two input sequence files produced by JOB0 and the cached DB sequence file. It performs the search using BMH algorihm and produces a report with the scan results.

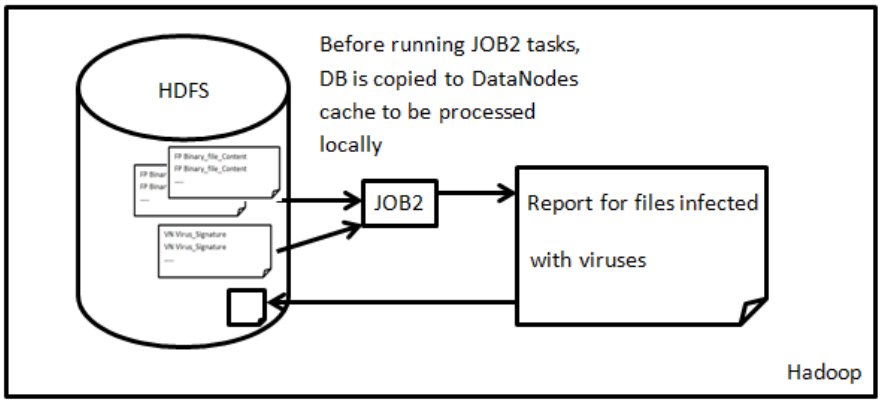
The performance of the system *(Table3)* on pseudo-distributed mode was checked using Cloudera Manager and Hadoop Tracking Interface. A Snapshot of searching phase which is JOB2 is shown in *(Fig8)*.

Table 3. System Performance Using Sequence files and BMH Searching for 10 Portable Executable Files

JOB ID	JOB Description	Elapsed Time	INPUT	OUTPUT
JOB0	PE-files preparation phase	40secs	10 files	2 Input Sequence files
JOB1	AV-signature preparation phase	33secs	9 DB files	1 DB Sequence file
JOB2	Searching PE files for viruses and reporting results phase	2mins:14secs	2 Input Sequence files and cached DB Sequence file	Report with scan results



a) A Detailed System Architecture illustrating JOB0 and JOB1



b) A Detailed System Architecture illustrating JOB2

Fig. 7.

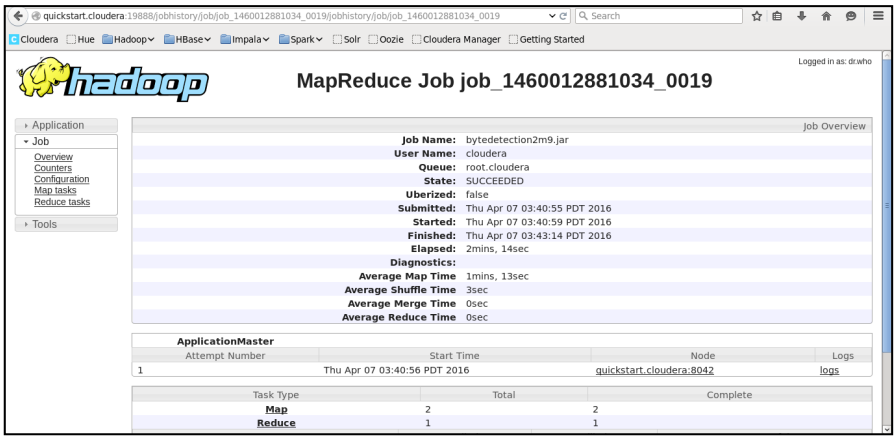


Fig. 8. Performance of System Searching Phase

7 Running on Virtual Multi-node Cluster

In this section, the application, done above, was tested on a virtual multi-node cluster. This cluster consists of three virtual machines: n1, n2 and n3. The machines were built using Cloudera CDH4 and VMware workstation. The machines use centos 64-bit as guest operating system. CM 5.4.0 is used to trace executions. RAM: n1 (8GB, 1CPU), n2 (2GB, 1CPU), n3 (2GB, 1CPU).

- For HDFS layer: n1 (namenode, secondary namenode), n2, n3 (datanodes).
- For MapReduce Layer: the job is running using MapReduceV2 *(Fig10)*: n1 (resource manager), n2 (nodemanager), n3 (nodemanager) *(Fig9)*. YARN has a single MapReduce JobHistory server that holds the tracing history of the jobs executed in this cluster. Usually, the job history server runs on the same node as the resourcemanager. n1 is the master in this cluster and has the JobHistory.

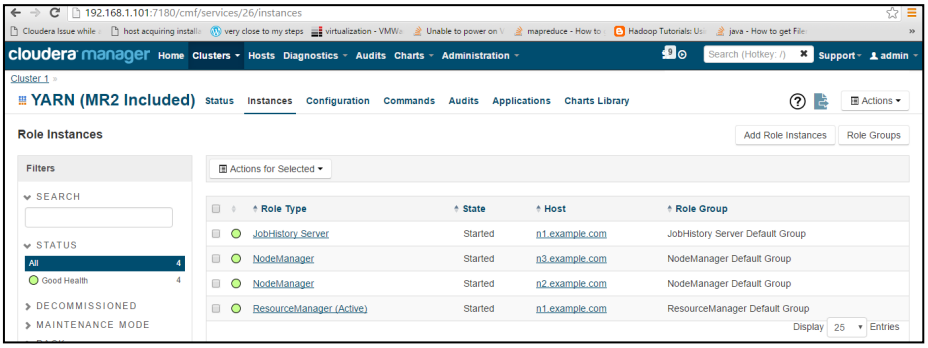


Fig. 9. Yarn Instances in the Virtual Multi-node Cluster

N.B.: the container is JVM used for processing. The resourcemanager (RM) is global manager for all applications (jobs) in the system. One of nodemanagers (NM) will be allocated as applicationmaster (AM). AM works per-application (per-job). This AM-job will be tasked using containers of other NM(s) allocated by resourcemanager. AM cooperates with all other NMs to execute and monitor the running tasks (4).

In this virtual cluster: the nodemanager n3 was chosen by RM n1 to be AM and jobs were tasked in the nodemanager n2 *(Fig11)*.

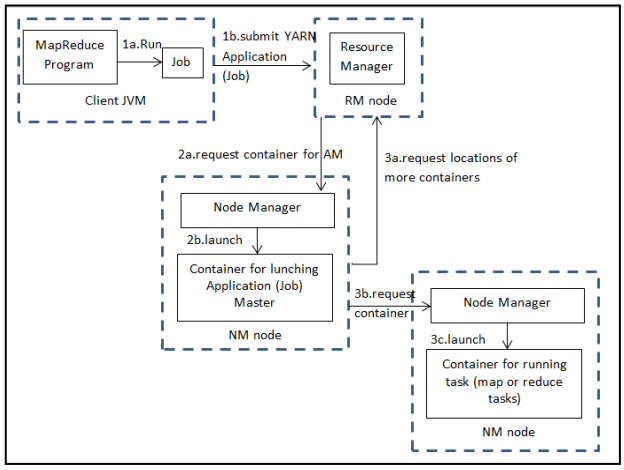


Fig. 10.YARN Running Architecture

The performance of the system *(Table4)* on virtual cluster was checked using Cloudera Manager and Hadoop Tracking Interface. A Snapshot of searching phase which is JOB2 is shown in *(Fig11)*.

Table 4. System Performance on the Virtual Multi-node Cluster

JOB ID	JOB Description	Elapsed Time	INPUT	OUTPUT
JOB0	PE-files preparation phase	1min:25secs	10 files	2 Input Sequence files
JOB1	AV-signature prepa-ration phase	1min:17secs	9 DB files	1 DB Sequence file
JOB2	Searching PE files for viruses and reporting results phase	2mins:44secs	2 Input Sequence files and cached DB Se-quence file	Report with scan results

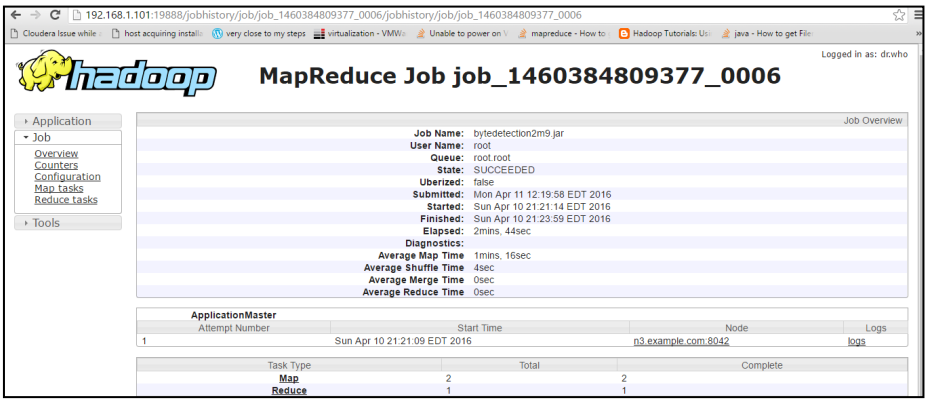


Fig. 11. Performance of System Searching Phase

A comparison between the two running modes: pseudo-distributed and virtual multi-node cluster for the application is shown in the next figure *(Fig12)*.

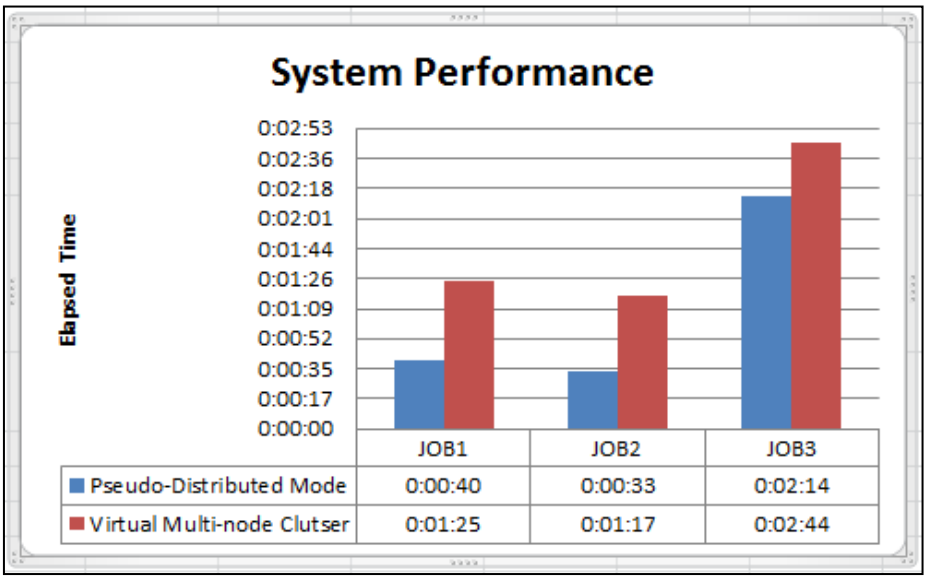


Fig. 12. Pseudo-Distributed Vs. Virtual Multi-Node Cluster

From the above results, the virtual multi-node cluster elapsed a little bit more time, because of the limited physical resources used as described previously: n1 (8GB RAM, 1CPU), n2 (2GB RAM, 1CPU), n3 (2GB RAM, 1CPU). However, in pseudo-distributed mode, the ClouderaquickstartVM was running at (8GB RAM, 2CPU) which helped in running tasks faster than the cluster. This shows that if the system is applied in a real cluster with more physical resources, it might show an increase in the performance compared to these two modes.

Even in the pseudo-distributed mode still there are limitations in the number of samples used because it is not powerful as real machines on real networks. Having limited physical resources, when trying to increase the samples to 20 samples a RAM error appears which demands using another machine with higher RAM and this will be carried in future work on a real network. Hence, increasing number of samples demands increasing network hardware resources.

The advantage of this architecture that it can prepare both types of files input files and DB before starting the scan. It utilized static search algorithms on cloud environment while up to our knowledge in previous works either it was not handled on cloud environment or it was introduced in cloud environment but using preexisting antivirus tools. Moreover the presented architecture, utilizes the advantages of Hadoop in organizing and speeding up the search in the cluster nodes.

8 Conclusion and Future Work

From the previous discussion, handling static search with Hadoop environment can be done by overcoming small files' problem. Searching speed is enhanced by using BMH algorithm. DB location is determined depending on system requirements to avoid bottlenecks. A simple tool was used to form the infected executable files. ClamAV DB was used in the scanning. The system was done on several steps to test the different factors affecting performance. The previous discussion described how to overcome these factors during testing and proved a noticeable increase in performance. Although the testing was done to serve infected files scanning, it can be utilized in other fields as solving the problem of handling and processing of small files in Hadoop as in image processing. This paper presented a simple way of a better understand of how to scan infected files using static search across Hadoop platform.

On-going is, bringing this system to real cluster. To this point the system was tested on Hadoop pseudo-distributed mode which is very close to what is happening on a real cluster. Furthermore, it was tested on multimode cluster of three virtual machines to see how the application runs in hadoop cluster, but because of limited physical resources, still there is a need to try it on real cluster.

9 References

[1] Malware Analysis Using Hadoop and MapReduce. NK Dengle, SC Dharmadhikar. s.l. : AVCOE, Sangamner, 2015. Fourth Post Graduate Conference.

[2] Binarypig: Scalable static binary analysis over hadoop. Hanif, Zachary, Telvis Calhoun, and Jason Trost. s.l. : Black Hat USA , 2013.

[3] MRSI: A fast pattern matching algorithm for anti-virus applications. Zhou, X., Xu, B., Qi, Y., & Li, J. s.l. : IEEE, 2008. Networking Seventh International Conference . pp. 256-261.

[4] Hadoop. https://hadoop.apache.org/. [Online] accessed [April-2016].

[5] WordPress. https://learnhadoopwithme.wordpress.com/tag/hadoop-daemons/. [Online] accessed [April-2016].

[6] European Institute for Computer Anti-Vir Research. INTENDED USE. http://www.eicar.org/86-0-Intended-use.html. [Online] accessed [April-2016].

[7] ClamAV. https://www.clamav.net. [Online] accessed [April-2016].

[8] Virus Total. https://www.virustotal.com. [Online] accessed [April-2016].

[9] String Matching Methodologies: A Comparative Analysis. Rasool, Akhtar, et al. s.l. : REM (Text) 234567.11, 2012.

[10] Practical fast searching in strings. Horspool, R. Nigel. s.l. : Software: Practice and Experience 10.6, 1980, pp. 501-506.

[11] Improving Speed of the Signature Scanner using BMH Algorithm. Kanaujiya, Sunita, S. P. Tripathi, and N. C. Sharma. s.l.: International Journal of Computer Applications 11.4, 2010.

[12] Amazon. Import files with Distributed Cache. http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-plan-input-distributed-cache.html. [Online] accessed [April-2016].

10 Authors

Eman Ahmed (corresponding author) is with Ain Shams University, Cairo, Egypt (e.ah.saad@gmail.com).

Amin Sorrour (corresponding author) is with Misr University for Science and Technology, 6th of October, Egypt (asorrour@yahoo.com).

Mohammed Sobh is with Ain Shams University, Cairo, Egypt (mohamed.sobh@eng.asu.edu.eg).

Ayman Bahaa-Eldin is with The British University in Egypt and on leave from Ain Shams University, Cairo, Egypt (ayman.bahaa@bue.edu.eg).

This article is a revised version of a paper presented at the BUE International Conference on Sustainable Vital Technologies in Engineering and Informatics, held Nov 07, 2016 - Nov 09, 2016 , in Cairo, Egypt. Article submitted 26 December 2016. Published as resubmitted by the authors 23 February 2017.