IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 10, Issue 3 (Mar. - Apr. 2013), PP 33-42
www.iosrjournals.org

Resource Allocation for Antivirus Cloud Appliances

Ali Abdullah Hamzah, Sherif Khattab, Salwa S. El-Gamal
Department of Computer Science, Faculty of Computers and Information
Cairo University, Egypt

Abstract: *Malware detection or antivirus software has been recently provided as a service in the cloud. A cloud antivirus provider hosts a number of virtual machines each running the same or different antivirus engines on potentially different sets of workloads (files). From the provider's perspective, the problem of optimally allocating physical resources to these virtual machines is crucial to the efficiency of the infrastructure. This paper proposes a search-based optimization approach for solving the resource allocation problem in cloud-based antivirus deployments. An elaborate cost model of the file scanning process in antivirus programs is instrumental to the proposed approach. The general architecture is presented and discussed, and a preliminary experimental investigation into the antivirus cost model is described. The cost model depends on many factors, such as total file size, size of code segment, and count and type of embedded files within the executable. However, not a single parameter of these can be reliably used alone to predict file scanning time.*
Keywords: *Cloud computing, virtualization, antivirus, pattern matching*

I. Introduction

Cloud computing allows for the management and provisioning of resources (e.g., software, CPU, memory, I/O, network bandwidth, application, and information) as services provided over the internet on demand. Services are presented in three layers: SaaS (software as a service), such as Face book and YouTube, PaaS (platform as a service), such as Microsoft Azure and Google AppEngine, and IaaS (infrastructure as a service), such as Amazon EC2 and GoGrid [1]. Cloud computing is based on virtualization, which abstracts physical computer resources and allows users to create and run multiple virtual machines (VMs) on a single physical machine, whereby each virtual machine is allocated a share of the physical machine resources. The virtual machines can potentially run different operating system with different applications [2].

Antivirus (AV) software is one of the most important applications in information technology to prevent and remove malware infection. The effectiveness of traditional host-based AV is limited on the long term in detecting many modern threats for many reasons, such as the time window from the time a virus's signature is released to the time the signature is delivered to the AV running on a device, and the high resource requirement, which may make mobile device users opt for stopping their antivirus [3].

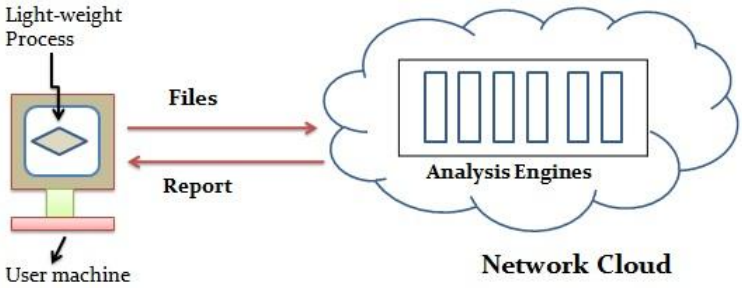

Figure 1: Cloud antivirus architecture.

Providing antivirus as a service in the cloud, such as the CloudAV model [4], addresses the above-mentioned limitations. As shown in Fig.1, a cloud antivirus provider hosts one or more malware analysis engines and provides its service to remote clients over the internet. Files are sent from the clients to the cloud, get scanned, and the result is sent back to the client. The concept of a cloud antivirus is simple: instead of running complex analysis software on every end-host, each end-host runs a lightweight process. This lightweight process discovers potential threats to the system and sends them to a network-accessible service for analysis.

This paper investigates the problem of resource allocation in a CloudAV-like setting. A cloud antivirus provider hosts a number of virtual machines, each running the same or different antivirus engines processing different workloads (files). Lightweight processes on end-hosts collect and upload suspect files to the

cloud for scanning. The performance of the cloud antivirus infrastructure is affected by the resource allocation to the virtual machines. Optimizing resource allocation would improve resource utilization.

The contribution of this paper is two-fold: (1) a general search-based framework for solving the resource allocation problem in cloud antivirus setting is presented and (2) an instrumental part of the framework, namely the antivirus cost model, is investigated experimentally to model the file scanning time. The experimental results show that scanning time depends on many factors, such as total file size, size of code segment, and count and type of files embedded within the scanned executable. However, none of these parameters alone was found to reliably estimate the scanning time.

The rest of this paper is organized as follows. Section II describes background on virtualization, antivirus software, and pattern matching algorithms. In Section III, the problem of resource allocation in cloud antivirus environment is presented. Section IV describes the proposed solution, namely the Automatic Cloud Antivirus Configurator (ACAC). In Section V, the cost model is experimentally investigated. Section VI discusses closely related research, and the paper is concluded by Section VII, which also briefly presents future work.

II. Background

His section presents a brief background on virtualization, the corner stone of cloud computing, and a more detailed background on the theoretical underpinnings of antivirus performance.

A. *Virtualization*

The idea of virtualization is to separate user's perception of software and hardware from the physical resources. Virtualization adds a layer of software between applications and physical resources used by these applications. This layer maps virtual resources perceived by applications to the real physical resources. In addition, it allows for multiple applications to share common physical resources [2]. One of the famous resource virtualization types is machine virtualization. Machine virtualization technology depends on software called the virtual machine monitor (VMM). Examples of VMM software include Xen, VMware, and Virtual box [5].

B. *Pattern Matching Algorithms*

This subsection gives more detailed background on pattern matching algorithms, which are used in signature-based antivirus programs. In particular, we discuss the WO pattern matching algorithms that are used in ClamAV [6], the open-source antivirus program that is used as a case study in this work. For each algorithm, we describe how it operates and its asymptotic running time.

AV software plays an important role in today's internet communication security. AV is used for scanning emails, files being transferred, and files on disk. The scanning speed of an AV program is important to optimize the speed of the whole scanning process. AV scanning speed relies on the pattern matching algorithm that is implemented in the AV software. Antivirus software uses WO main strategies for detecting viruses and other malware: signature-based and behavior-based detection.

Signature-based detection is the traditional way of antivirus operation. It works by matching patterns stored in a database using pattern matching algorithms, such as Boyer Moore [7] and Aho-Corasick [8].

Behavior-based detectors monitor behavior of all programs and can detect unknown and new viruses. In addition to the run-time behavior, static characteristics can also be determined to reinforce the identification of malicious behavior. Algorithms used include support vector machines (SVM) [9] and decision tree [10].

Pattern matching algorithms have recently been applied to many applications, such as internet security, antivirus, search engines, and data retrieval. In pattern matching we have a pattern *pat* of length *m*, we have a string *str* of length *n*, where usually $n > m$, and we want to find the positions (in *str*) of all occurrences (if any) of *pat* in *str* [11]. The focus of this paper is on the use of pattern matching algorithms in antivirus (AV) applications.

Among the various famous pattern matching algorithms in AV software (ClamAV as a case study) are the Boyer-Moore algorithm [7] and the Aho-Corasick algorithm [8], which are explained in more detail below.

1) Boyer-Moore algorithm*:* The general idea behind the Boyer-Moore (BM) algorithm well be presented [7]. BM was proposed for single-pattern matching. It works by sliding the pattern from left to right over the text and starting the comparison from right to left. BM employs three heuristics in the comparison as follows: (1) Right to left comparison: the comparison operation in BM algorithm starts from right-most character of the pattern with corresponding character in the text and second character of rightmost of pattern with corresponding character in the text and so on as shown in Fig. 2.

(2) The bad character heuristic works in WO steps. **First,** when the comparison of the rightmost character in the pattern with the corresponding character in the text gives a mismatch and the mismatched character in the text does not appear in the pattern, and then the pattern is moved as a whole to the right. For

example, in Fig. 2(b), we can observe that the character *F* of the text has a mismatch with the rightmost character *T* of the pattern. In addition, *F* does not occur in the pattern, and thus, the pattern is moved as a whole to the first character after *F* (the character A).

Second, when the comparison gives a mismatch and the mismatching character in the text matches a character elsewhere in the pattern, the pattern is moved to the right until the two matching characters are aligned. In Fig. 2 (c), we can observe that the character (-) of the text has a mismatch with the character *T* of the pattern. In addition, (-) occurs in the pattern, and thus, the pattern is moved to the right until the pattern character aligns to the same character of text (e.g., the character (-) of pattern aligns to the character (-) of text)

(3) In the good suffix heuristic, after a mismatch, when a suffix of more than one character in the pattern matches the corresponding substring of text and is found to occur in another position in the pattern, then the algorithm slides the pattern to the right until the matching substring in text aligns to the other occurrence of the pattern suffix. For example, in Fig. 3(a), we can observe that the substring of the two rightmost characters (AT) of the pattern match the corresponding characters of the text, but the third character from right is a mismatch (H in pattern and '-' in text). In the same time, we observe that the substring AT is found in the first two positions of pattern. Thus, the pattern is slid until its substring AT is aligned to the substring AT in the text (Fig. 3(b)).

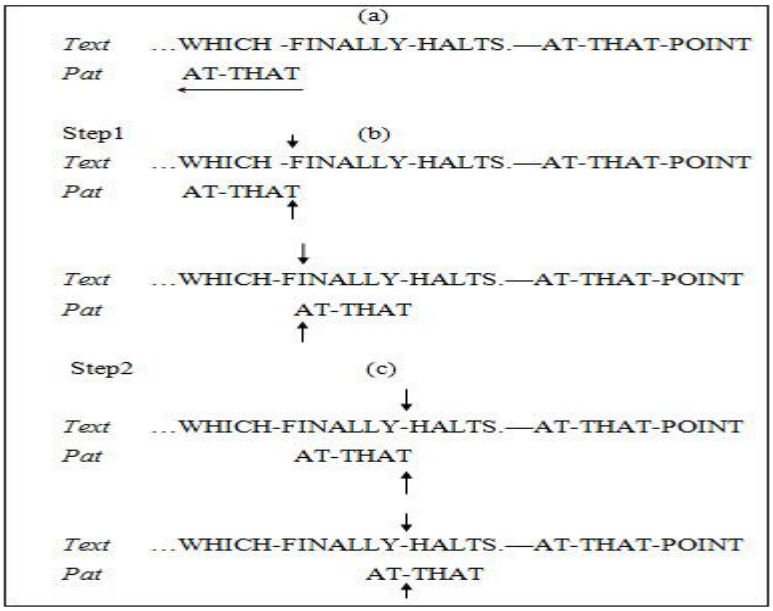

Figure 2: The Boyer-Moore algorithm (bad character heuristic).

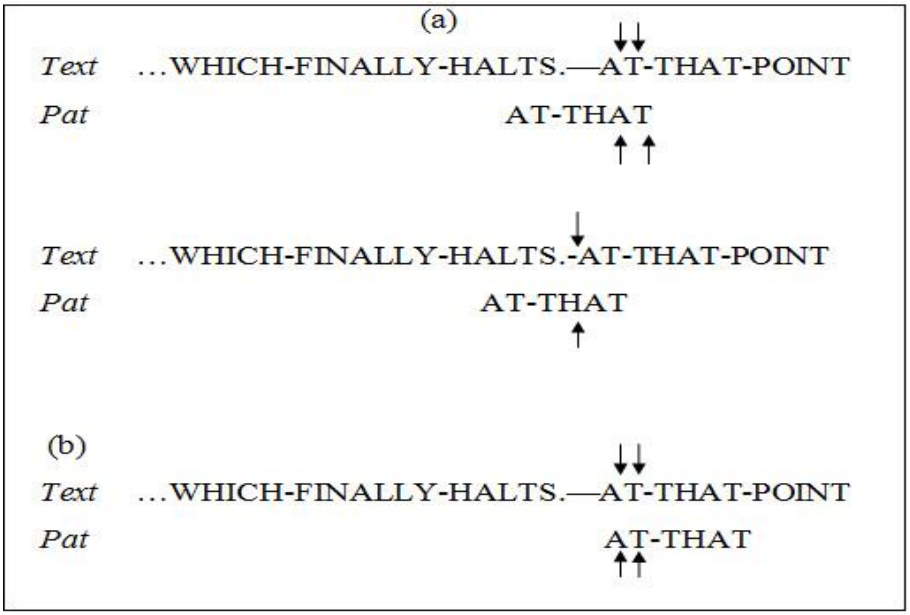

Figure 3: The Boyer-Moore algorithm (good suffix heuristic).

***2) Aho-Corasick algorithm (AC)* [8]**: For multi-pattern matching, the AC algorithm constructs a finite state automaton that accepts all strings in the pattern. The automaton is built in the preprocessing stage. The AC algorithm operates in *O(n)* running time. It includes the following functions:

(a) The Goto Function

The automaton starts in the start state (state 0). When the first input symbol is accepted, the *goto* function *g ()* is applied on the pair (state 0, input symbol) and the machine transitions into the next state. In other words, the next state becomes the current state and the next input symbol the current symbol. If the input symbol is not accepted, then a fail event happens. For example, *g*(7,d)= fail because no pattern contains substring "hisd" (Fig. 4).

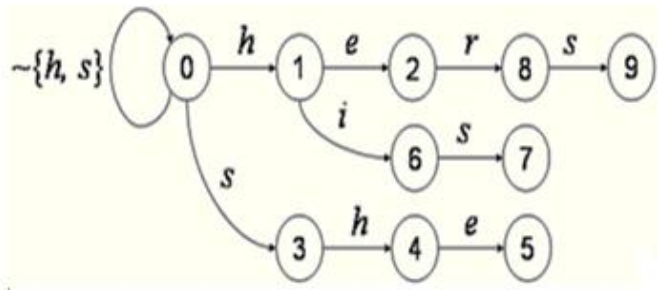

Figure 4: The *goto* function of the Aho-Corasick algorithm.

(b) The failure function

Failure function *f*( ) is called when *g*(state, input symbol) = fail, that is, the input symbol is not expected in the current state. In other words, if the string resulting from the concatenation of the labels of the transition arcs from the starting state to the current state plus the input symbol is not a prefix of any pattern, the machine makes a transition to failure by calling the failure function, and it repeats the above-mentioned operation starting from the failure state. In Figure 5, in state 7, the machine returns to state 3 when it receives an unexpected symbol.

s	1	2	3	4	5	6	7	8	9
F(s)	0	0	0	1	2	0	3	0	3

Figure 5: The failure function of the Aho-Corasick algorithm.

(c) Output function

The output function *O* ( ) maps each state into a set of patterns (could be empty), which contains a pattern string *p* if *p* is a suffix of any pattern in the state machine. As an example, we have *O (5) = {she, he}*. Note that the symbol (e) is a suffix of the patterns she and he (Fig. 6).

s	***output(s)***
2	***{he}***
5	***{she,he}***
7	***{his}***
9	***{hers}***

Figure 6: The output function of the Aho-Corasick algorithm.

III. Resource Allocation Problem

This section describes the problem of allocating physical resources to virtual machines running antivirus software in a cloud computing setting.

Virtualization is a common enabling technology in cloud computing. In cloud computing, the virtualization technology is used to share resources among virtual machines that run on physical servers. Resource allocation is one of the problems faced in order to optimize performance of the applications running inside the virtual machines. These applications are called *cloud appliances*. Deciding on how many physical resources to allocate to each virtual machine not only affects the performance of cloud appliances but also the efficiency of the underlying physical infrastructure.

Application performance is affected by the allocation of physical resources to the VMs. Different applications demand different resource allocations, and moreover, different workloads within the same application pose different resource requirements. For example, The performance of some applications and

workloads is optimized by allocating more CPU to their appliances, whereas other applications and workloads benefit more from higher memory or I/O allocation. This heterogeneity motivates the need to control and configure resource allocation to the virtual machines.

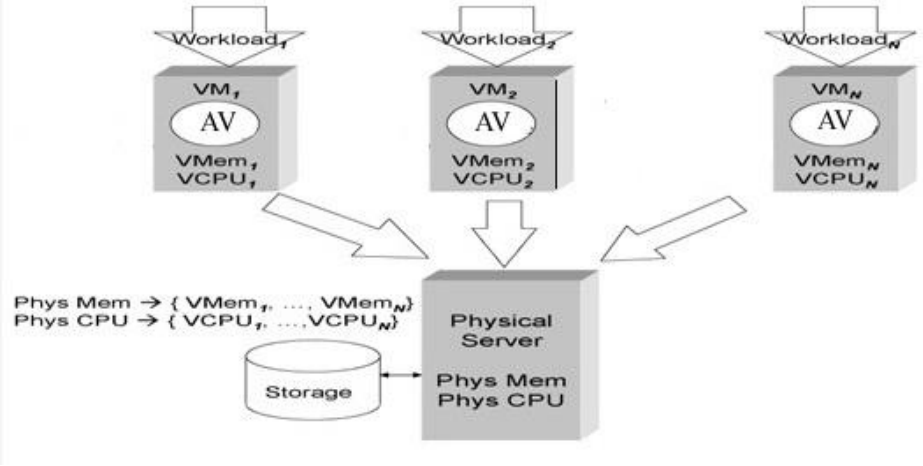

Fig.7. Resource allocation for cloud antivirus appliances

As shown in Fig.7, *N* VMs run on a physical machine. Each VM runs an antivirus program in a cloud antivirus. ClamAV is used as an example of an antivirus [5]. The shared physical resource (e.g., CPU and memory) is represented by *R*. Each VM has a workload, which is a set of files to scan. These files can be clean or virus-infected. *Wi* represents the workload on each VM, and r_{ip} represents the CPU allocation to Vim. This work focuses on the allocation of the CPU resource over virtual machine. The CPU allocation to the VMs is represented as a vector $[r_i]$. For example, for three VMs, the first taking 50% of CPU, the second 30%, and the third 20%, the allocation is [0.5, 0.3, 0.2].

A resource allocation is better than another if it yields less cost. The cost in this work is measured as the time needed to scan the input files assigned to a VM. The total cost is simply the total cost over all VMs. Estimating the cost of scanning a file is instrumental to reaching an optimal or near-optimal allocation.

IV. Automatic Cloud Antivirus Configurator (ACAC)

This section describes our proposed solution to the resource allocation problem presented in the previous section.

The proposed solution is based on a search algorithm that enumerates possible resource allocations and evaluates them based on a cost model. The cost model provides the search algorithm with an estimate of the CPU time needed to scan the workload files on a particular VM with a given CPU allocation. The parameters of the cost model may need to be adjusted in order to reflect the virtual environment under the provided CPU allocation. This adjusting process is called *parameter calibration.*

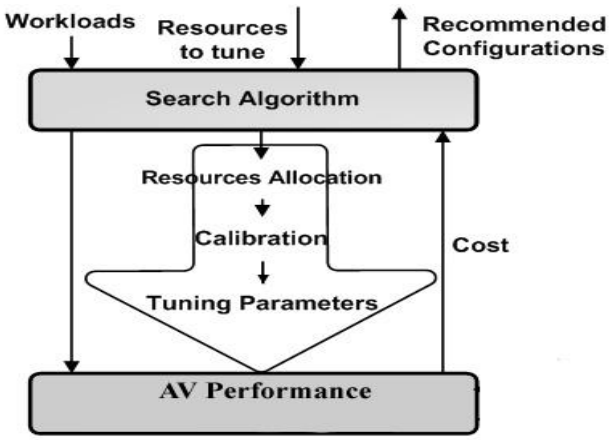

Fig. 8: The Automatic Cloud Antivirus Configurator (ACAC).

Fig.8 depicts the architecture of the proposed solution, namely the Automatic Cloud Antivirus Configurator (ACAC). ACAC is divided into three modules: the configuration enumerator, which includes the

search algorithm, the parameter calibration process, and the AV cost model, which estimates the cost scanning a set of files under a given CPU allocation. This architecture is inspired by the Virtual Design Advisor [12], which has been proposed to solve the resource allocation problem in a different context.

Configuration enumerator (search algorithm): The configuration enumerator module is used to enumerate resource allocations for the VMs. It implements a search algorithm, such as greedy search and dynamic programming, for enumerating candidate resource allocations. An example greedy algorithm [12] initially assigns a 1/N share of each resource to each of the *N* VMs. It then proceeds iteratively. In each iteration, it considers shifting a share (say, 5%) of some resource from one VM to another. The algorithm considers all such resource reallocation, and if it finds reallocation of resources that is beneficial according to the cost estimator, then it makes the most beneficial reallocation and iterates again. Basically, in each iteration, a small fraction of a resource is deallocated from VM that will get hurt the least and allocated to the VM that will benefit the most. If no beneficial reallocations are found, the algorithm terminates, reporting the current resource allocation as the recommended allocation.

Calibration process: Calibration is a process used for mapping each candidate resource allocation to a set of parameter values in the antivirus cost model. In the cost model, a set of parameterized equations are used to compute a cost estimate. Some of the parameters may need to be adjusted according to the resource allocation to the virtual machine. The calibration process adjusts these parameters before applying the equations.

AV performance model: ClamAV [6] is an open-source antivirus toolkit for UNIX. The main purpose of it is e-mail scanning on email gateways. It is the most widely used open-source antivirus and is based on pattern matching algorithms (e.g., signature-based). ClamAV uses signature-based strategy in the scanning process, whereby it uses two algorithms, namely multi-pattern Boyer-Moor and Aho-Corasick. Theoretically, the running time for the single-pattern Boyer-Moore algorithm is O(*n/m)* and for Aho-Corasick is O (*n*), where *n* is the text size and *m* the pattern size.

In practice, however, there is more than one parameter that effect the cost, such as the number and type of file types embedded within a file and the number of signatures targeted for each file type. Fig. 9 shows some of the inputs to the cost model, which is an instrumental part of the proposed architecture. The next section experimentally digs deeper into understanding the cost model of an antivirus program.

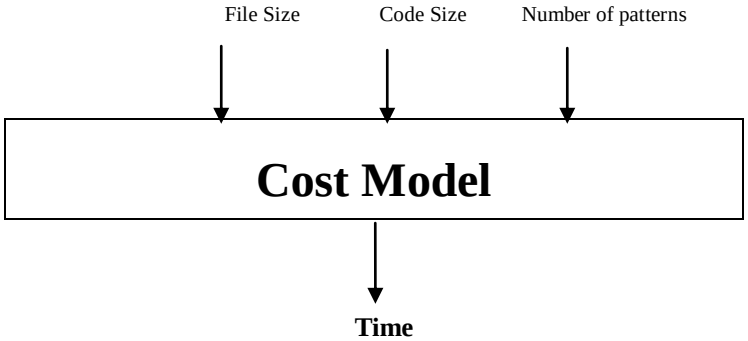

Figure 9: The antivirus performance model.

V. Antivirus Cost Model

This section presents the results of a number of experiments that explore the effect of some of the parameters that affect the performance of an antivirus.

A. *Experiment setup*

We used ClamAV version 0.97.3 installed on a machine with Core 2 Duo™ T5870 2.0 Processor, 2 GB Memory and Centos 5.5 operating system. We used the Xen platform as the virtualization environment. Xen is an open source virtualization software, and it is used in its par-virtualization mode [13]. The experiment used two types of workload: a set of clean files with file size between 1 and 25 MB and a set of known malware with size between 1 and 10 MB. Both the clean and infected files were Windows executables. The clean files were

grouped into five groups based on file size as follows: 1 MB to 5MB, 5MB to 10MB to15MB, 15MB to 20 MB, and 20MB to 25MB. Similarly, the malware files were grouped into three groups: 0 MB to 1MB, 1MB to 5MB, and 5 MB to 10 MB. Each file was scanned ten times with averages reported.

To automate scanning, the following simple script was used:

```
#! /bin/csh
foreach file ( `ls |  xargs)
  echo –n  > / logs/$file.exe.log
  foreach is (1 2 3 4 5 6 7 8 9 10)
    clamscan $file.exe > /logs/$file.exe.log
  end
end
```

B. *Effect of file size*

The first experiment examined the effect of total file size on scanning time. Inspired by the linear theoretical cost model, a linear regression analysis was conducted between file size and scan time. The linear regression yields an equation:

$$y = A*x + B \qquad (1)$$

where A is the slope of the line, and B is the y-intercept.

Fig. 10 shows the linear regression analysis at a CPU allocation of 100%, and Table 1 shows the result of the regression analysis for the set of clean files at CPU allocation of 100%, 70%, 50%, and 30%. The table shows the values of the linear equation parameters ($y = A*x + B$) at each CPU allocation. *It is important to note that some files with larger size took less time than files that are smaller in size.*

Table 1: Effect of file size on scan time at different CPU allocations

CPU allocation	A	B
100%	0.224	3.796
70%	0.692	11.009
50%	1.425	20.191
30%	2.463	38.021

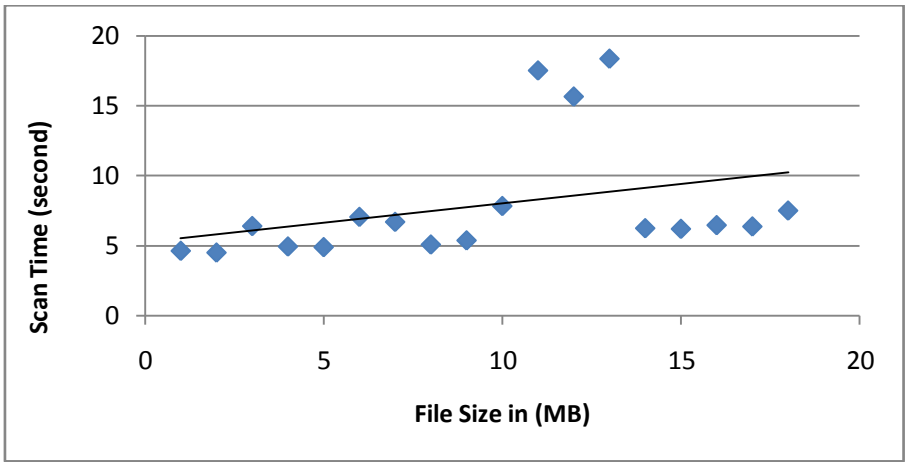

Figure 10: Linear regression analysis of scan time vs. file size for clean files at 100% CPU allocation.

Fig. 11 shows the results of the linear regression analysis for the set of malwares at a CPU allocation 100%.

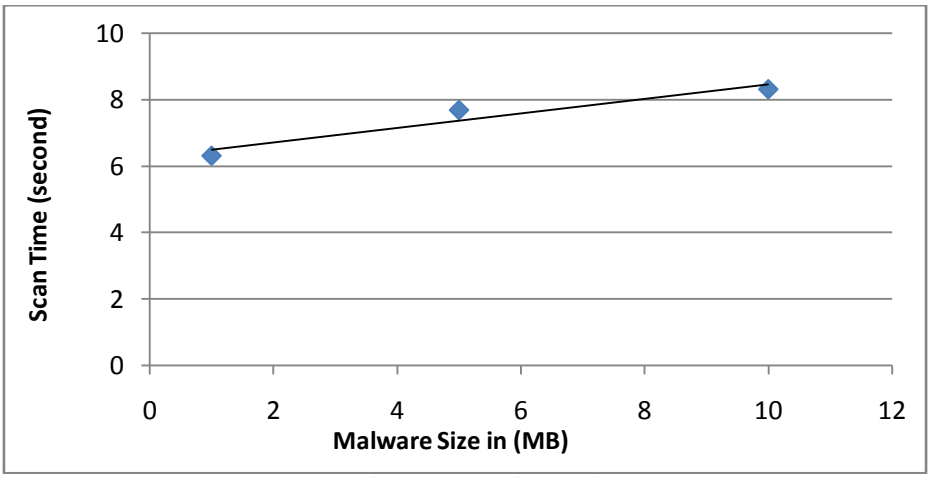

Figure 11: Linear regression analysis of scan time vs. file size for malware at 100% CPU allocation.

C. *Effect of code size*

Through observation from the previous experiment, the scan time for a file did not necessarily depend on the file size. In the second experiment, only the size of the code segment of the file is used as an indicator of the file's scan time.

To extract the code size, the PE explorer software [14] was used. Fig. 12 depicts the linear regression model for scan time vs. code size. It can also be noted that the code size is not a reliable indicator of scan time if used alone. *Some files with smaller code size took longer to scan than files with smaller code size.*

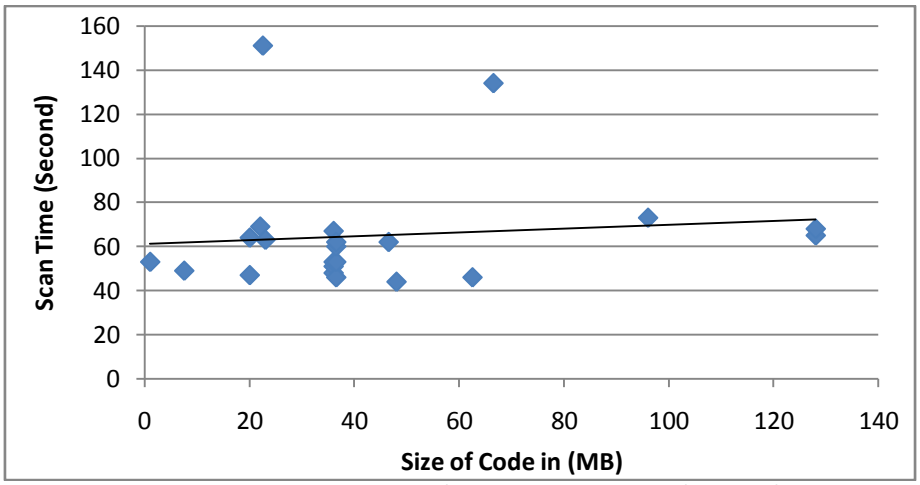

Figure 12: Linear regression analysis of scan time vs. code size for clean files.

D. *Effect of number of embedded types*

The third experiment investigated the effect of the number of file types embedded within a Windows executable. Some executables contain embedded files within, such as archives and compressed files, documents, mail formats, PDF files, and others. The count and type of the embedded files has an effect on the scan time, because some of these files are decompressed or extracted and scanned, and each type invokes a potentially different set of signatures to check against. Types such as HTML, ARCHIVE, PE, OLE2, MAIL, ELF, and ASCII affect the scan performance because each type has a specific number of patterns in the ClamAV database. However, as shown in Fig. 13 the number of embedded types alone is not a reliable indicator.

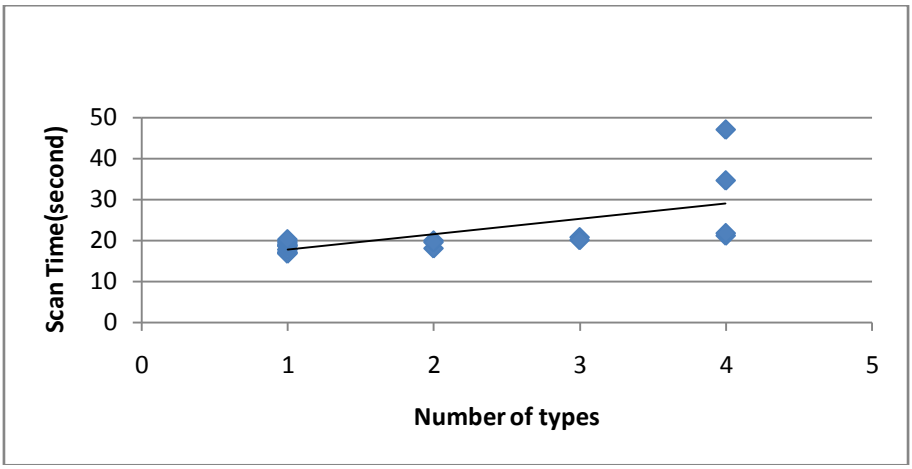

Figure 13: Linear regression analysis of scan time vs. number of embedded type for clean files.

To neutralize the effect of embedded files, ClamAV was run while removing the processing of all embedded types. Fig. 14 shows that the scan time is reliably linearly dependent on the scanned size, which matches the theoretical performance.

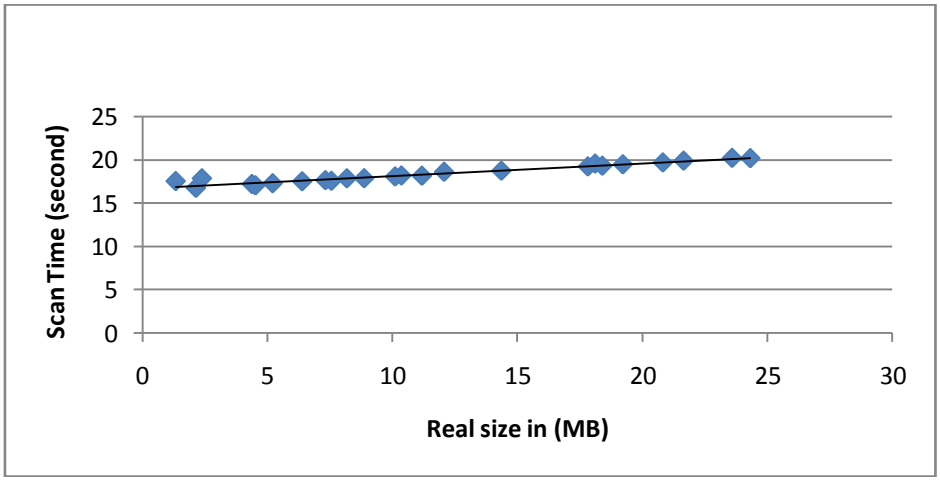

Figure 14: Linear regression analysis of scan time vs. scanned size after removing all embedded file processing

VI. Related Work

Performance optimization at the IaaS cloud layer has been the subject of active research recently. For example, the virtualization design problem was addressed in the database community [12], [15]. The authors addressed the problem of resource allocation over virtual machines running database management systems. The virtual machine configurations determine how the shared physical resources will be allocated to the different database instances. The implemented solution, namely the virtual design advisor, indirectly uses information about the anticipated workloads of each of the database via a cost model, the query optimizer, to estimate the effect of a particular resource allocation on workload performance.

Traditional antivirus software has to deal with new threats and an increasing surface of vulnerabilities on the long run. Hence, an antivirus must be scalable and flexible for new deployments. Particularly in cloud computing, malware detection is provided as a service in the cloud. This setting is particularly suitable for mobile devices through moving the antivirus functionality off-device to the network and employing multiple virtualization malware detection engines [4].

In CloudAV [4], a lightweight process at the client sends files to a server in the cloud for scanning, allowing for integrated antivirus software, behavioral simulation, and other deep-analysis engines from multiple vendors providing better detection of malware [16].

A novel anti-malware system has been proposed and called SplitScreen. SplitScreen performs an additional screening step prior to the time- and bandwidth consuming process of downloading all virus signatures to a client. Non-infected files are processed faster, and possibly infected files are scanned only against a small set of potential signatures [17]. A similarity between our approach and SplitScreen is the preprocessing of files to make the subsequent scanning process more efficient. However, the preprocessing setup is done for different specific objectives. In our work, preprocessing gives an estimate of the scan time to

aid allocation problem, whereas in SplitScreen, processing aids to download only the necessary signatures to scan a particular file or set of files.

VII. Conclusions and Future Work

A cost model of antivirus programs is instrumental to the approach adopted in this paper. An accurate cost model would guide the search algorithm to optimal or near-optimal resource allocation. Our preliminary experiments showed that the scan time (i.e., the time taken by an antivirus program to scan a file) depends on the total file size, the size of the code segment, and the count and types of embedded files inside the executable. However, not a single parameter of these can be reliably used alone to estimate the scan time. Hence, in the future, a machine-learning approach taking all these parameters as features well be investigated.

References

[1] R.Qi Zhang, Lu Cheng, "Cloud computing: state of the art and research challenges," in *J. Internet and Applications.* The Brazilian Computer Society 2010, pp. 7-18.

[2] Virtualization [online]. Available at http://www.vmware.com/virtualization/. Last checked on April 14, 2013.

[3] S. K. Shah and N. I. R., "Exploring Reliability of cloud antivirus solution," in *New Jersey Institute of Technology.*

[4] J. Oberheide, E. Cooke, and F. Jahanian, "CloudAV: N-version antivirus in the network cloud," in *Proceedings of the 17th conference on Security Symposium* SS'08, Berkely, CA, USA: USENIX Association, 2008, pp. 91-106.

[5] R. Rose, "Survey of system virtualization techniques," Tech. Rep. 2004.

[6] Clam antivirus [online]. Available at http://www.clamav.net. Last checked on April 14, 2013.

[7] R. S. Boyer and J. S. Moore, "A fast string searching algorithm," *Commun. ACM*, vol. 20, no. 10, pp. 762-772, Oct. 1977.

[8] A. V. Aho and M. J. Corasick, "Efficient string matching: an aid to bibliographic search," *Commun*. ACM, vol. 18, no. 6, pp.333-340, Jun. 1975.

[9] Bo-yun Zhang, Jian-ping Yin, Jin-bo Hao, Ding-xing Zhang, and Shu-lin Wang, "Using support vector machine to detect unknown computer viruses," *International Journal of Computational Intelligence Research., vol. 2*, no. 1, pp. 100-104, 2006.

[10] D. J. Hand, P. Smyth, and H. Mannila, "*Principles of Data mining."* Cambridge, MA, USA: MIT Press, 2001.

[11] P. S. WHEELER, "Techniques for improving the performance of signature-based network intrusion detection system," Master's thesis, Wake Forest University, CALIFORNIA, 2003.

[12] A. Soror, U. F. Minhas A. Aboulnaga, K Salem, P. Kokosielis, and S. Kamath, "Automatic virtual Machine configuration for database workload," In proc. ACM SIGMOD *International Conference On Management of Data (SIGMOD'08),* Jun.2008, pp. 953-966.

[13] Introduction to the Xen virtual machine monitor [online]. Available at http://www.linuxjournal.com/article/8540?page=0,1. Last accessed on April 14, 2013.

[14] View, Edit, and Reverse Engineer EXE and DLL Files [online]. Available at: http://www.heaventools.com/overview.htm. Last accessed on April 14, 2013.

[15] A. Soror, U. F. Minhas A. Aboulnaga, K Salem, P. Kokosielis, and S. Kamath," Deploying database Appliances in the cloud," *IEEE Data Eng. Bull.*, Vol. 32, no. 1, pp. 13-20, 2009.

[16] J. Oberheide, E. Cooke, and F. Jahanian, "Rethinking antivirus: executable analysis in the network cloud," in *Proceedings of the 2nd USENIX Workshop, USA, 2007.*

[17] S. K. Cha, I. Moraru, J. Jang, J. Truelove, D. Brumley, and D. G. Andersen, Splitscreen: Enabling efficient, distributed malware detection," in *Proceedings of the 7th USENIX conference on networked systems.* USENIX Association, 2010, pp. 187-200.