BUILDING ACCURATE MODELS TO DETERMINE THE CURRENT CPU UTILIZATION OF A HOST WITHIN A VIRTUAL MACHINE ALLOCATED ON IT

Samira Briongos

LSI - Electronic Engineering Department
Universidad Poliécnica de Madrid, Spain
CCS-Center for Computational Simulation
Madrid, Spain
samirabriongos@die.upm.es

Pedro Malagón

LSI - Electronic Engineering Department
Universidad Poliécnica de Madrid, Spain
CCS-Center for Computational Simulation
Madrid, Spain
malagon@die.upm.es

José L. Risco

Department of Computer Architecture and Automation
Complutense University of Madrid
Computer Science Faculty, Spain
jlrisco@ucm.es

José M. Moya

LSI - Electronic Engineering Department
Universidad Poliécnica de Madrid, Spain
CCS-Center for Computational Simulation
Madrid, Spain
josem@die.upm.es

ABSTRACT

In cloud computing environments there are several virtual machines running in the same host. This fact opens the door for possible side channel-attacks. Prior to perform an attack it is mandatory to determine co-residency with the victim. An synchronized variation in the CPU activity in the host is a possible indicator of the presence of neighboring processes. However, cloud providers do not give information about the hosts CPU load, so we have to figure out a way of estimating it. We estimate the host CPU load considering its impact on the performance of a virtual machine (VM) running on it. In this work, we show that it is possible to calculate the CPU load of the host by executing a reference process, measuring the time it takes to execute, and using this information as an input to generate the CPU load models. We explore regression methods and regression methods tuned with genetic algorithms for the model generation. As a result, considering a CPU load value between 0 (no load) and 100 (maximum load), we obtain models which compute the host load with a mean squared error of around 5%, 10% and 30% (depending on the host architecture) when estimating the load every second.

Keywords: Cloud Computing, side-channel, Genetic-algorithm, LASSO

1 INTRODUCTION

Since its inception, cloud computing has experienced an exponential growth. Big companies such as Amazon, Microsoft or Google have become cloud computing providers, with both government and industry using public cloud platforms as clients. Cloud computing enables on demand access to computing resources, allowing users to adapt their computing capabilities to their demands and eliminating for them the need to acquire and maintain expensive IT infrastructures. In this scenario, where cloud customers pay for the re-

sources they use, several virtual machines (VMs) run in the same host. This sharing of physical resources between tenants allows cloud providers to achieve economies of scale, reducing the operations and maintenance cost of IT infrastructures.

The isolation between VMs running on the same physical machine is provided by a virtual machine manager (VMM) and supervised by a hypervisor. However, the underlying hardware creates covert channels between VMs or processes, that can be exploited to break the logical isolation between tenants. Side-channel leakage can be used to extract private information from co-resident VMs (Yarom and Falkner 2014, Irazoqui, Inci, Eisenbarth, and Sunar 2014) such as keys from cryptographic algorithms, as well as to detect the execution of processes like *sshd* in co-resident VMs (Suzaki, Iijima, Yagi, and Artho 2011). For all these side-channel attacks to succeed, there is one requisite: co-residency. Prior to the attack, a potential attacker has to determine if the target is allocated on the same host or not.

There have been several sucessful attempts to detect co-allocation. In 2009, the work from Ristempart et al. (Ristenpart, Tromer, Shacham, and Savage 2009) paved the way for the upcoming works in co-residency detection and cross VM side-channel attacks. They proved that co-residency is achievable and detectable in the Amazon's Elastic Compute Cloud (EC2). Although the techniques they used to determine the placement of the victim, have been obfuscated by Amazon and other cloud providers, new techniques are coming up (Inci, Gulmezoglu, Irazoqui, Eisenbarth, and Sunar 2015).

In this work, we consider the problem of co-residency detection from a different perspective. When a process or VM is running, it consumes CPU cycles, modifying the CPU utilization of the host. This suggests that there must exist a correlation between task executions on the potential victim virtual machine, and changes in the CPU load of the host. However, cloud providers do not facilitate information about host utilization to their clients. Users can only access usage statistics related to their own virtual machines. Thus, we have to devise a technique to gain information about the host utilization. The main objective of this work is the development and validation of accurate models of the current CPU utilization of a host which can be obtained within a VM running on it.

When a process shares the CPU with others, its execution time varies. In this work we use this variation to estimate the load of the host. Our previous work (Briongos, Malagón, Risco-Martín, and Moya 2016) suggests that this relation may be linear, so we first employ regression techniques (LASSO) to obtain the model. In a second approach, we use Genetic Algorithms to tune the inputs of the LASSO in order to get simpler models with similar error rates. In summary, we show that it is possible to obtain accurate models of the CPU load within a process or VM running on it, by just executing and measuring a test process. Our models estimate the load of the host with mean squared errors (MSE) below 28% when calculating the CPU load from a process running native on the physical machine, and below 38 % when the calculations are performed inside a virtual machine (worst case, LASSO regression).

The remaining paper is organized as follows. First, we overview the related work on the area in section 2. Next, we present our proposed method to obtain the models, altogether with the algorithms used, in section 3. Then, we describe our experimental setup (section 4) and show the most relevant results in section 5. Finally, we draw some conclusions in section 6.

2 RELATED WORK

By definition, cloud computing shares the same physical machine between multiple clients (VMs), increasing the utilization ratio of the machine and, as an indirect outcome, creating co-residency data leakage problems. For example, CPU caches, which are shared between all the CPU cores and consequently between all the VM running in a host, can be used to extract information from a neighbour VM. They have been used to infer AES (Irazoqui, Eisenbarth, and Sunar 2015), RSA (Liu, Yarom, Ge, Heiser, and Lee

2015), EDCSA (Yarom and Benger 2014) and other cryptographic keys. Zhang et al. (Zhang, Juels, Oprea, and Reiter 2011) used the L2 cache side-channel with defensive purposes. They build a tool able to detect co-resident foe VMs. However, the techniques they described can be adapted and used to detect co-resident "victim" machines.

Zhang et al. (Zhang, Juels, Reiter, and Ristenpart 2014) who demonstrated attacks between tenants on commercial Platform-as-a-Service (PaaS) clouds, also studied the attacker's ability to co-locate an instance with a victim. The strategy they described consists of two steps: first, the adversary has to launch a large number of instances on the cloud service, and second, he attempts to determine somehow whether any of the instances co-locates with the victim instance. We suggest using CPU utilization as a mean of co-residency detection, but as a previous step we need to prove that it is possible to estimate the current CPU load of the host machine within an instance. That is, in this work we build the CPU utilization models which can potentially be used to determine co-residency.

Masti et al. (Masti, Rai, Ranganathan, Müller, Thiele, and S.Capkun 2015) found that, in modern multi-core platforms, the isolation techniques based on dedicated cores can be circumvented using thermal channels. They were able to detect processes which were executed in neighboring cores by monitoring the temperature sensors of the cores and quantifying the heat propagation as an effect of running CPU intensive applications. In a similar way, our work studies the effect that running different loads in a host has in a test process.

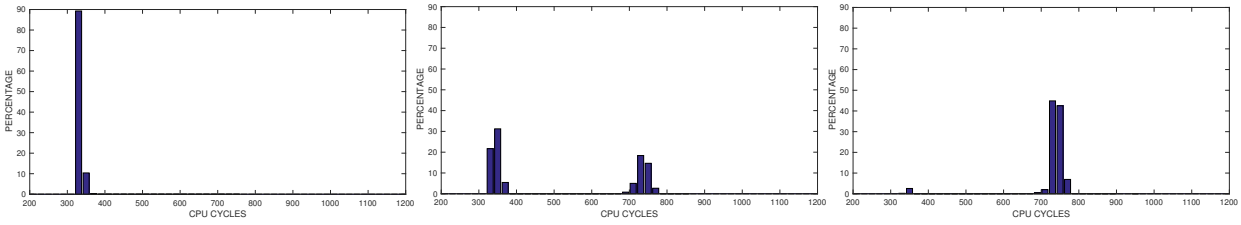
Additionally, Liu (Liu 2011) used a well-known relation between CPU load and temperature to estimate the load of the host in a cloud environment, also by reading temperature sensors. Their objective was conducting a study of server utilization in public clouds. Thus, a model of CPU utilization, as the one we propose, can be used to estimate utilization in public clouds. Regarding to temperature sensors, they have to be exposed by the hypervisor in order to be read inside the VMs, and they are not always exposed. In fact, we have tried to read them using KVM as hypervisor, with the default configuration settings, and we have not been able to. In contrast, our approach makes use of time counters, which are available inside VMs, to calculate the total load of the host.

Adams et al. (Adams, Bello, and Dumancas 2015) use Genetic Algorithms to reduce the number of input features for different modeling algorithms, including *LASSO*. They select 24 relevant questions from a questionnaire with 123 questions.

3 PROPOSAL

When a process is executed simultaneously with other processes in the same physical machine, they share the CPU. As a consequence, execution times of the same process vary between executions. We propose to execute in a loop a reference process and measure its execution time. We have selected AES encryption as our reference process. It is a high CPU consumer process, it is quite fast (we can perform several encryptions in milliseconds) and, additionally, the time it takes to perform an encryption is almost constant. When monitoring, we execute encryptions one after the other and measure the time it takes to finish each of them. Figure 1 depicts different histograms of the encryption time under different conditions of CPU load. We define the CPU load as the number of cycles the CPU was executing some task divided by the number of passed cycles (considering all the cores), and multiplied by 100. As a consequence, the CPU load will take values between 0% and 100%.

If we analyze Figure 1, it seems that the number of encryptions lasting around 350 CPU cycles decreases linearly with the increase of activity in the host machine. When there is no load in the host, around 100% of the encryptions are in this interval, while there are almost no encryptions in the mentioned interval when the host CPU load is close to 100% . Taking as an input the distribution of the times it takes the processor to perform an encryption, we try to fit a linear model to calculate the host CPU load using a linear regression



(a) No load (b) 65% load (c) 98% load

Figure 1: Encryption CPU cycles (time) histograms under different CPU loads

techniques, *LASSO*. For this step, we select the limits of the histograms between 100 cycles and 1700 in steps of 20 cycles; that is, we have 80 inputs for the *LASSO* algorithm. To reduce this number, and still be able to calculate the load of our host, we combine the *LASSO* with a Genetic Algorithm. The goal is then finding the limits which, with the minimum number of inputs, provide the minimum error in the estimation. We finally propose using piecewise functions for the model generation when the error of the model generated is grouped around a concrete CPU load. Therefore, we can generate multiple models. The selection of the model to be used is done according to the input value of the histogram.

For a better understanding of our work we provide a brief background on this techniques.

3.1 Least absolute shrinkage and selection operator

Tibshirani proposes the least absolute shrinkage and selection operator algorithm (*LASSO*) (Tibshirani 1994) that minimizes residual summation of squares according to the summation of the absolute value of the coefficients that are less than constant.

The algorithm combines the favourable features of both subset selection and ridge regression like stability, and offers a linear, convex and derivable solution. *LASSO* provides interpretable models shrinking some of the coefficients and setting others to exactly zero values for generalized regression problems.

For a given non-negative value of λ, the *LASSO* algorithm solves the following problem:

$$\min_{\beta_0,\beta}\left(\frac{1}{2N}\sum_{i=1}^{N}(y_i-\beta_0-x_i^T\beta)^2+\lambda\sum_{j=1}^{p}|\beta_j|\right) \tag{1}$$

where:

- β: vector of p components. *LASSO* algorithm involves the L^1 norm of β
- β_0: scalar value.
- N: number of observations.
- y_i: response at observation i.
- x_i: vector of p values at observation i.
- λ: non-negative regularization parameter corresponding to one value of Lambda. The number of nonzero components of β decreases as λ increases.

3.2 Genetic Algorithms

Genetic Algorithms (GA) (Koza 1992) are a method for solving optimization problems based on natural selection. A GA requires a population of candidate solutions (individuals or chromosomes), coded with a genotype. The genotype is a list of integer values, that are called genes, that can be translated to the desired solution format, which is the phenotype. With a fitness function the algorithm calculates the quality of every individual solution of the population. The GA starts with a population of individual solutions to the problem. Iteratively, it creates a new generation, a set of new candidate solutions, from the previous one with three operators:

- Selection: the candidates that best fit the problem are selected for the next generation (lower value of the fitness function).
- Crossover: two candidate solutions (parents) are selected between the best candidates to generate a new candidate by mixing their chromosomes. The child is generated with sequence of genes from both parent.
- Mutation: altering one or multiple genes of a chromosome to generate a new candidate solution.

The configuration of the GA includes the genotype, the fitness function definition, the number of individuals in each generation, the fraction of new candidate solutions generated with each operator and the number of generations to be executed to find the best solution.

The goal of our GA is to reduce the number of input features of the model generated using *LASSO*. The candidate solutions represent different histogram configurations, considering the width of each time interval.

At the end, we combine the use of GA that generates the better histogram codification in terms of input feature generation *LASSO* that computes the coefficients and the independent term in the final linear model.

As a result, our GA+*LASSO* framework solves our optimization problem that targets the generation of accurate CPU load models for host machines from execution time histograms.

4 EXPERIMENTAL SETUP

In this section we describe the scenarios in which we have performed our experiments, and detail the processes of gathering the data for obtaining our models, and the techniques employed.

4.1 Scenarios

We consider two different scenarios to evaluate our approach: virtualized and non-virtualized environments. For each of the scenarios we run our experiments in three different intel machines, each with different number of cores:

- Intel Xeon CPU E3-1226 v3 (3.30GHz), 4 cores.
- Intel Core i7-4790(3.60 GHz), 4 cores with two threads per core.
- Intel Xeon CPU E5-2620 (2.00GHz), 6 cores with two threads per core.

The OS of all the machines is Centos, and the hypervisor for all our experiments in virtualized environments is KVM. In all our VMs we have also Centos installed. The instances have 2 virtual CPUs (VCPU), although only one would be enough as we are only running one process, the monitor. We have selected machines

with different number of cores as the minimum load observed in our samples is the load we generate with our test process, which is around *(1/Number of cores)x100*. That is, our test process uses almost one core during its execution.

4.2 Data acquisition

In both scenarios (non-virtual and virtual), the way of collecting the data is similar. There are two main processes involved in the procedure of gathering the data we use to train and test our models:

- The process we monitor to evaluate the effects of the load on it. It consists on an AES encryption operation, from the OpenSSL library. During a fixed period of time, one second in our experiments, we perform one operation after another and measure the time it takes to perform each one. To obtain the times we use the `rdtsc` instruction, which returns the current value of the processor's timestamp in cycles. It is available inside the VMs. At the end of each period, we have an array with the times of each encryption performed and a counter with the number of encryptions that have been performed. We construct an histogram with the number of encryption operations whose duration is between two time values separated by 20 CPU cycles, starting in 100 cycles and ending at 1700 cycles. The total number of time divisions is consequently 80, which are the inputs of our models. Additionally, we store a timestamp tag with the previously computed histogram, obtained with the function `gettimeofday`.
- The process in charge of collecting information about the real use of the host. We collect from the host the values of the CPU load by reading the `/proc/stat` file once a second. We obtain the busy and idle cycles of each core of CPU in the last time window by subtracting the previous value from the current one. Then, we calculate the CPU utilization as the percentage between busy cycles and total cycles. As in the previous case, we store the value of the CPU utilization with its corresponding timestamp.

Additionally we generate different CPU loads in the host using lookbusy (Carraway), a synthetic load generator. Lookbusy is an application for generating load on a Linux system. It makes a system as busy as specified. Load can be induced at a fixed level or on a repeating cycle, allowing the definition of different load profiles. For example, it is possible to imitate a daily traffic load curve in a server. The lookbusy application adjusts its own consumption up or down to compensate other loads on the system.

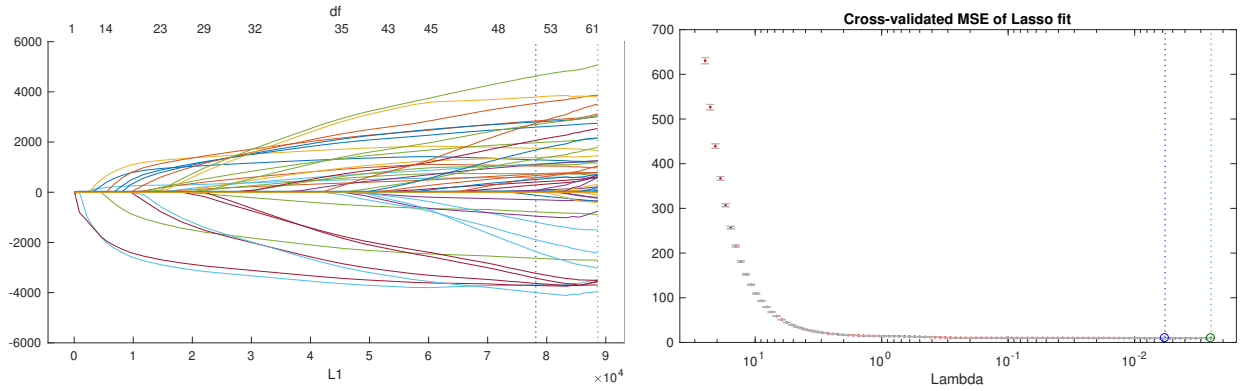
Once we have all the data, and before using it as an input for the regression algorithm, we have to match each sample of the histogram with its corresponding value of the real CPU load. To do so, we compare the timestamps of the data collected with the two processes and assign each histogram with a load value.

4.3 Methodology

4.3.1 LASSO

We use the Matlab software (MATLAB 2015) to perform the regression using *LASSO* algorithms. We call the function `lasso(X,Y,'CV',10)` where the predictors in X are the values of the histogram and the responses in Y are the values of the CPU load. With this call, we also indicate Matlab to use K-fold cross validation to estimate the mean squared error (MSE) of our model, and the K value is set to 10. As a result of this call, we obtain the coefficients of the fit, and their associated errors. In other words, we have our model of the host CPU load as a function of our encryption times histogram. One example of the results

obtained, can be seen in figure 2. Figure 2 also shows that the number of required inputs to compute our model is high, while the error rates are reasonable. That is the reason we decided to tune our inputs with a GA.



(a) Coefficients fit by LASSO

(b) MSE error

Figure 2: Outputs of the *LASSO* regression (10-fold cross validation)

4.3.2 LASSO tuned with GA

The Matlab software also includes a toolbox that allows the use of customized GA. We have used this toolbox to define our population individuals or phenotypes, a fitness function and the mutation and crossover functions which will be used to obtain the next generation.

Population: we consider 100 individuals per generation. The chromosomes of each individual represent the width of the time divisions of the histogram. With this data we are able to reconstruct our histograms as if the time limits were placed in different positions.

Fitness function: The objective of a GA is to minimize the fitness function. Based on the initial value of the MSE (the error obtained when fitting the model with the *LASSO* with the histogram unmodified) we define the fitness for each individual as the MSE obtained after performing the *LASSO* with the recalculated histograms, plus a penalization term given by the number of the significant variables for our model if the new MSE is close to the initial MSE. On the contrary we define our fitness as the MSE multiplied by the number of significant variables of the model. This way the best individuals are those whose MSE is similar to the original with the minimum number of inputs.

Next generation: the next generation is obtained as follows: 5 individuals are the "elite" and remain unchanged, the roulette wheel selection algorithm selects the parents for the next generation, 80 children will be obtained through crossover and the remaining children trough mutation. In the crossover step two parents are combined to generate one child, we randomly select one point of the phenotype and "cut" both parents at it. Next the child is obtained recombining one part of each parent. We define a mutation rate of 1%, that is, each child obtained though mutation has only one gen, also randomly selected, different than its parent.

The GA is executed during 500 generations, which takes a mean time of about 3.5 hours for each model. In the following section we present the results for each approach.

5 RESULTS

For each of the machines tested during our experiments, we present a table summarizing the results in terms of MSE, standard error (SE) and number of significant features in non-virtualized and virtualized environments, for each of the two approaches: *LASSO* and *LASSO* tuned with GA.

	Non-virtualized			Virtualized		
	MSE	SE	Features	**MSE**	SE	Features
Lasso	9.22	0.206	49	8.59	0.144	61
Lasso + GA	10.48	0.065	1	9.89	0.215	2

Table 1: MSE, SE and number of features in non-virtualized and virtualized environments for the Xeon E3

The models for CPU load estimation as a function of the histogram weights h_i, for the Xeon E3 machine are:

$$CPULoad(Nonvirt) = 27.6 + 39.45 * h_5 \tag{2}$$

$$CPULoad(Virt) = 103.45 - 2.24 * h_1 + 35.15 * h_3 \tag{3}$$

In this situation, the GA has been really useful to simplify our model. The errors are slightly bigger, but not statistically significant. We have also tried to apply the model obtained for non-virtualized environments to estimate the CPU in virtualized environments. The result is a MSE of 51.12, but even in this situation we are able to detect changes in the CPU load. This means that the model will still be useful to detect the activity of neighboring processes.

	Non-virtualized			Virtualized		
	MSE	SE	Features	**MSE**	SE	Features
Lasso	4.19	0.037	51	4.47	0.009	59
Lasso + GA	4.53	0.026	6	8.39	0.411	2

Table 2: MSE, SE and number of features in non-virtualized and virtualized environments for the i7

The equation for CPU load estimation in the i7 machine are:

$$CPULoad(Nonvirt) = 71.93 - 0.65 * h_2 + 0.63 * h_4 + 0.78 * h_5 + 6.72 * h_6 - 0.38 * h_8 + 0.62 * h_{10} \tag{4}$$

$$CPULoad(Virt) = 60.08 - 0.46 * h_2 + 120.95 * h_4 \tag{5}$$

In this case there is a noticeably difference between virtualized and non-virtualized environments. In both scenarios the number of input features required to estimate the load of the host has been reduced significantly by using the GA. However, in virtualized environments the error has increased from 4.47 to 8.39 while in non-virtualized environments it remains similar. We have also tried to interchange the models. In this case we obtain a higher MSE (58), but we are still able to notice changes in the CPU load.

	Non-virtualized			Virtualized		
	MSE	SE	Features	**MSE**	SE	Features
Lasso	27.79	0.565	56	37.76	0.327	76
Lasso + GA	33.24	0.112	6	56.61	1.12	4

Table 3: MSE, SE and number of features in non-virtualized and virtualized environments for the Xeon E5

The resulting models for CPU load estimation for the Xeon E5 machine are:

$$CPULoad(Nonvirt) = 16.34 - 0.25 * h_2 + 4.18 * h_3 + 1.55 * h_6 + 0.43 * h_7 + 1.56 * h_8 - 81.46 * h_9 \quad (6)$$

$$CPULoad(Virt) = 71.45 - 0.65 * h_2 - 0.60 * h_4 + 0.03 * h_5 + 1757.6 * h_7 \quad (7)$$

This Xeon E5 machine is the one which obtains the worst results, in both virtualized and non-virtualized environments. As a difference with the previous machines, we had one background process collecting information about the frequency of the CPU. We have observed several changes in the frequency while we were gathering the data for our models. As this machine had a different governor managing the frequency of the cores, this results suggest that it may be interesting to include the frequency in our models.

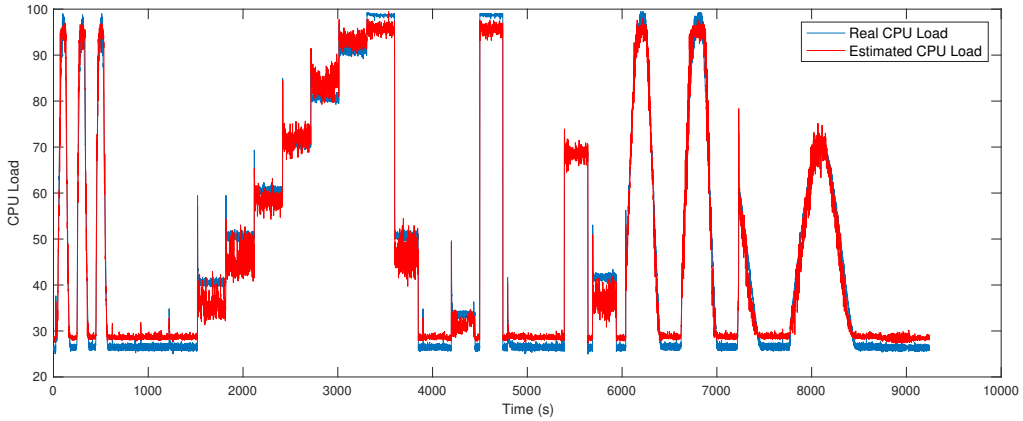


Figure 3: CPU real and estimated load

Figure 3 shows the values of the real and estimated CPU load of the host for a virtualized environment on the Xeon E3 machine. This data belongs to a different set than the data used for training our models. We can see in that our estimated load is unable to reach the minimum load of the host. This suggests that, if we are able to distinguish between the situation when there are other processes executing rather than the test process and when there are not, we could build a different model. In fact, it is possible to distinguish both situations easily in the Xeon E3 machine. We have determined an empirical threshold taking into account only the values of h_5 and h_3. When these values are below the threshold, the test process is "alone". We have used the real load associated with this values to train a new model using the *LASSO* and as a result our new models are:

$$CPULoad(Nonvirt) = \begin{cases} 24.85 + 53.11 * h_5 & h_5 < 0.08 \\ 27.6 + 39.45 * h_5 & other \end{cases}$$

$$CPULoad(Virt) = \begin{cases} 124.75 - 3.42 * h_1 + 28.02 * h_3 & h_1 > 31 \\ 103.45 - 2.24 * h_1 + 35.15 * h_3 & other \end{cases}$$

In the case of non-virtualized environments the MSE drops to 8.91. In the virtualized environment it also drops to 8.16. In both cases the results obtained are better than the previous cases.

6 CONCLUSIONS

In this work we have presented a discussion about the utility of having accurate models to estimate the CPU load of a server in cloud computing environments. We propose to detect co-allocation of virtual machines (VMs) by estimating the host CPU load and triggering an event in the victim VM. A synchronized variation in the load of the host with the trigger can be used as indicator of both VMs being executed in the same physical machine. We have proved that it is possible to derive the CPU load of a server by measuring the time it takes a process to execute n times during a fixed period of time. The histogram with the number of executions for multiple time intervals are input features for our modeling algorithms. First we use LASSO algorithm to obtain a linear regression model of the CPU load from the histogram data. Second, we tune the algorithm by using a Genetic Algorithm to select the best histogram intervals for the model. The model generates a linear formula that uses the values of the histograms as inputs and outputs the estimated host CPU load. We ave selected AES encryption as the reference process. It tends to use one full core for itself. Therefore, the minimum CPU load of our traces, and consequently of our models, is determined by the number of cores. As a consequence, we have performed our experiments in different machines with different CPU cores. The MSE of our estimation depends on the scenario considered and on the physical machine. Our results vary between a MSE of 4.19 in the best case and 56.61 in the worst case. Based on our results, we have re-generated our models as a piecewise function, which outperforms our initial results. For future modeling algorithms it may be interesting to consider Grammatical Evolution algorithms to obtain our CPU utilization models. To sum up, in this work, we have laid the basis to try our approach in a public cloud, and to use it as part of a co-residency detection algorithm.

ACKNOWLEDGEMENTS

This project has been partially supported by the Spanish Ministry of Economy and Competitiveness, under contracts RTC-2014-2717-3, TIN-2015-65277-R, AYA2015-65973-C3-3-R and RTC-2016-5434-8.

REFERENCES

Adams, L. J., G. A. Bello, and G. G. Dumancas. 2015. "Development and Application of a Genetic Algorithm for Variable Optimization and Predictive Modeling of Five-Year Mortality Using Questionnaire Data". *Bioinformatics and Biology Insights* (Suppl 3), pp. 31–41.

Briongos, S., P. Malagón, J. L. Risco-Martín, and J. M. Moya. 2016. "Modeling Side-channel Cache Attacks on AES". In *Proceedings of the Summer Computer Simulation Conference*, SCSC '16, pp. 37:1–37:8. San Diego, CA, USA, Society for Computer Simulation International.

Devin Carraway. "Lookbusy: a synthetic load generator". https://www.devin.com/lookbusy/. Accessed: 2017-04-16.

Mehmet Sinan Inci and Berk Gulmezoglu and Gorka Irazoqui and Thomas Eisenbarth and Berk Sunar 2015. "Seriously, get off my cloud! Cross-VM RSA Key Recovery in a Public Cloud". Cryptology ePrint Archive, Report 2015/898. http://eprint.iacr.org/2015/898.

Irazoqui, G., T. Eisenbarth, and B. Sunar. 2015, May. "S$A: A Shared Cache Attack That Works across Cores and Defies VM Sandboxing – and Its Application to AES". In *2015 IEEE Symposium on Security and Privacy*, pp. 591–604.

Irazoqui, G., M. Inci, T. Eisenbarth, and B. Sunar. 2014. *Wait a Minute! A fast, Cross-VM Attack on AES*, pp. 299–319. Cham, Springer International Publishing.

Koza, J. R. 1992. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. Cambridge, MA, USA, MIT Press.

Liu, F., Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. 2015, May. "Last-Level Cache Side-Channel Attacks are Practical". In *2015 IEEE Symposium on Security and Privacy*, pp. 605–622.

Liu, H. 2011, Dec. "A Measurement Study of Server Utilization in Public Clouds". In *2011 IEEE Ninth International Conference on Dependable, Autonomic and Secure Computing*, pp. 435–442.

Masti, R., D. Rai, A. Ranganathan, C. Müller, L. Thiele, and S.Capkun. 2015. "Thermal Covert Channels on Multi-core Platforms". In *24th USENIX Security Symposium*, pp. 865–880. Washington, D.C., USENIX Association.

MATLAB 2015. *version (R2015a)*. Natick, Massachusetts, The MathWorks Inc.

Ristenpart, T., E. Tromer, H. Shacham, and S. Savage. 2009. "Hey, You, Get off of My Cloud: Exploring Information Leakage in Third-party Compute Clouds". In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pp. 199–212. New York, NY, USA, ACM.

Suzaki, K., K. Iijima, T. Yagi, and C. Artho. 2011. "Memory Deduplication As a Threat to the Guest OS". In *Proceedings of the Fourth European Workshop on System Security*, EUROSEC '11, pp. 1:1–1:6. New York, NY, USA, ACM.

Tibshirani, R. 1994. "Regression Shrinkage and Selection Via the Lasso". *Journal of the Royal Statistical Society, Series B* vol. 58, pp. 267–288.

Yarom, Y., and N. Benger. 2014. "Recovering OpenSSL ECDSA Nonces Using the FLUSH+RELOAD Cache Side-channel Attack". *IACR Cryptology ePrint Archive* vol. 2014, pp. 140.

Yarom, Y., and K. Falkner. 2014. "FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack". In *Proceedings of the 23rd USENIX Security Symposium, San Diego, CA, USA, August 20-22, 2014.*, pp. 719–732.

Zhang, Y., A. Juels, A. Oprea, and M. K. Reiter. 2011. "HomeAlone: Co-residency Detection in the Cloud via Side-Channel Analysis". In *32nd IEEE Symposium on Security and Privacy, S&P 2011, 22-25 May 2011, Berkeley, California, USA*, pp. 313–328.

Zhang, Y., A. Juels, M. K. Reiter, and T. Ristenpart. 2014. "Cross-Tenant Side-Channel Attacks in PaaS Clouds". In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, pp. 990–1003. New York, NY, USA, ACM.

AUTHOR BIOGRAPHIES

SAMIRA BRIONGOS received a Degree in Telecommunication Engineering from the Technical University of Madrid, Spain, in 2014. Currently she is a Ph.D. Candidate in the Electronic Engineering Department at the Technical University of Madrid. She has been awarded a Research predoctoral grant of the Technical University of Madrid. She is a member of the HiPEAC European Network of Excellence. Her research interest include cloud computing security, particularly side-channel attacks and countermeasures and machine learning algorithms and their appliance to improve computer security. Her email address is samirabriongos@die.upm.es.

PEDRO MALAGÓN is currently an Assistant Professor in the Department of Electronic Engineering, Technical University of Madrid. He received his M.Sc. and Ph.D. degrees in Telecommunication Engineering from the Technical University of Madrid, Spain, in 2006 and 2015, respectively. He is a member of the HiPEAC European Network of Excellence and researcher at the Center for Computational Simulation. His research interest has focused on wireless sensor networks appliactions and security, including side-channel attacks. He is co-author of more than 30 publications on international journals and conferences. His current research interests focus on hardware implementations and machine learning algorithms, applied to security and performance. His email address is malagon@die.upm.es.

JOSÉ L. RISCO is currently an Associate Professor in the Department of Computer Architecture and Automation, Complutense University of Madrid. Previously he was Assistant Professor in the Computer Science Department at the Colegio Universitario de Segovia (Segovia, Spain) from 1998 to 2000, and Assistant Professor in the Computer Science Department at the C.E.S. Felipe II de Aranjuez (Aranjuez, Spain) from 2000 to 2006. He received the M.S. in Physics in 1998 at the Complutense University of Madrid, and his Ph.D. degree at the same University in 2004. He has served as paper reviewer, session chair, and conference organizer in different organizations. He is author of a book on netcentric system of systems engineering, another one on solved problems in digital design and over 100 articles in different venues. He has collaborated in the organization of different conferences in the area of modeling and simulation. His research interests are focused in the areas of Discrete Event Modelling and Simulation, parallel and distributed simulation, and Metaheuristics (with special focus in discrete optimization and feature engineering). His email address is jlrisco@ucm.es.

JOSÉ M. MOYA is currently an Associate Professor in the Department of Electronic Engineering, Technical University of Madrid. He received his M.Sc. and Ph.D. degrees in Telecommunication Engineering from the Technical University of Madrid, Spain, in 1999 and 2003, respectively. He is a member of the HiPEAC European Network of Excellence and the Spanish technological platforms eSEC and PROMETEO. He has served as the TPC member of many conferences, including DATE, IEEE ICC, IEEE MASS, IEEE GLOBECOM, DCOSS, etc. He has participated in a large number of national research projects and bilateral projects with industry, in the fields of embedded system design and optimization, and security optimization of embedded systems and distributed embedded systems. In these fields, he is co-author of more than 70 publications on prestigious journals and conferences. His current research interests focus on proactive and reactive thermal-aware optimization of data centers, and design techniques and tools for energy-efficient compute-intensive embedded applications. His email address is josem@die.upm.es.