19th Annual IEEE International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems

Workload Characterization at the Virtualization Layer

Fatemeh Azmandian
ECE Department
Northeastern University
Boston, USA
fazmandi@ece.neu.edu

Micha Moffie
ECE Department
Northeastern University
Boston, USA
mmoffie@ece.neu.edu

Jennifer G. Dy
ECE Department
Northeastern University
Boston, USA
jdy@ece.neu.edu

Javed A. Aslam
CS Department
Northeastern University
Boston, USA
jaa@ccs.neu.edu

David R. Kaeli
ECE Department
Northeastern University
Boston, USA
kaeli@ece.neu.edu

***Abstract*—Virtualization technology has many attractive qualities including improved security, reliability, scalability, and resource sharing/management. As a result, virtualization has been deployed on an array of platforms, from mobile devices to high-end enterprise servers.**

In this paper, we present a novel approach to working at a virtualization interface, performing workload characterization equipped with the information available at the virtual machine monitor (VMM) interface. Due to the semantic gap between the raw VMM-level data available and the true application behavior, we employ the power of regression techniques to extract meaningful information about a workload's behavior.

We also demonstrate that the information available at the VMM level still retains rich workload characteristics that can be used to identify application behavior. We show that we are able to capture enough information about a workload to characterize and decompose it into a combination of CPU, memory, disk I/O, and network I/O-intensive components. Dissecting the behavior of a workload in terms of these components, we can develop significant insight into the behavior of any application. Workload characterization can be used for online performance monitoring, workload scheduling, workload trending, virtual machine (VM) health monitoring, and security analysis. We can also consider how VMM-based workload profiles can be used to detect anomalous behavior in virtualized environments by comparing a model of potentially malicious execution to that of normal execution.

***Index Terms*—Workload Characterization, Virtual Machine Monitor, Linear Regression, Lasso**

I. Introduction

Virtualization has quickly grown in popularity due to its attractive benefits. Some of these features include: improved resource utilization, manageability, and energy savings in data centers and cloud computing, increased availability and reliability of IT services, fast deployment of services and infrastructure, and greater security of virtual machines (VMs). In this paper, we propose an added benefit of virtualization: *workload characterization* using the vantage-point of the Virtual Machine Monitor (VMM). In a virtualized execution environment, the VMM is a software layer that allows the multiplexing of different virtual machines on the shared physical hardware resources, each virtual machine running its own operating system (OS).

Working at the VMM-level, versus the VM-level or guest OS-level, significantly reduces the overhead introduced into the runtime by the profiling system. But raw VMM-level profile data provides limited program semantic information that is typically available at the application or operating system interfaces. We rely heavily on our ability to tap into the wealth of information that lies within the VMM-level data to reconstitute an accurate picture of the high-level program semantics.

Prior work has tackled this problem for the purpose of security and intrusion detection. Garfinkel and Rosenblum [1] utilize the VMM to provide virtual machine introspection (VMI), using their knowledge of the operating system structures inside the virtual machine to interpret VMM-level events into OS-level semantics. In a second study [2], [3], Moffie et al. demonstrate that VMM-level features utilized by sophisticated machine learning techniques can provide high intrusion detection accuracy with low false positive rates.

We also pursue this path, but investigate if we can utilize VMM-level raw events present in a number of well-behaved workloads to reconstruct the behavior of a real-world application. Again, we only consider information captured at the VMM-level. The extent to which a program interacts with the VMM can provide insight into the program's behavior. For example, a CPU-intensive program that mainly executes user-mode instructions should not require significant interaction with the VMM. For a program that is disk-intensive, the VMM will need to intercept requests to access the disk, triggering events that are unique to virtualized environments such as "enter recompiled execution mode (REM)".

The ability to reduce an application's behavior down to a *mix* of simpler behaviors can provide great insight into the performance and execution characteristics of an application. It also allows us to characterize the application and then detect abnormal activity caused by malicious execution.

Applications execution and behavior is composed of a number of different behaviors [4]: CPU-activity, memory-activity, and I/O-activity. As such, we begin by capturing the behavior of a set of workloads geared toward each of these components, which we refer to as our *canonical* workloads. For example, a CPU-intensive workload that focuses on pushing the limits of CPU activity comprises the CPU workload of our canonical set. We further distinguish I/O activity in terms of whether the activity is *disk* I/O-intensive or *network* I/O-intensive. Since I/O-intensive workloads are bidirectional (i.e., they perform

1526-7539/11 $26.00 © 2011 IEEE
DOI 10.1109/MASCOTS.2011.63

both reads and writes), we further decompose disk I/O-intensive workloads into disk *read*-intensive and disk *write*-intensive workloads and the network I/O-intensive workloads into network *receive*-intensive and network *transmit*-intensive workloads. Thus, our canonical workload set consists of the following workloads: CPU, memory, disk read, disk write, network receive, and network transmit.

For each workload in the canonical set, we capture low-level architectural data available at the VMM layer such as disk and network I/O accesses, page faults, and control register updates. Since we collect this at the VMM level, we can collect this information at one point for all VM's and identify unique processes in each VM. This raw data is then processed to produce a set of features on a per process or per VM basis. The features from each of the canonical workloads, along with the features of a target workload are provided as input to regression algorithms (the target workload is the workload being characterized). The regression algorithms produce a model of the target workload's activity as a linear combination of the activity of the canonical workload set. From the model, we can gain an understanding of the underlying behavior of the target workload. This model can aid in identification of performance bottlenecks, as well as improve the security of the virtual machine employing intrusion detection. The behavior of a suspected compromised machine can be compared against our new model of a workload (i.e., normal execution) to detect the presence of malware.

As part of our contributions, we have implemented a prototype of a VMM-level workload characterization tool using VirtualBox [5], an open-source full-virtualization VMM that runs on x86-based architectures. To the best of our knowledge, this is the first work to utilize *only* the low-level architectural information visible to the VMM for workload characterization.

Our tool consists of two key components:

- A *front-end*, whose capabilities include:
  - *Event Extraction* - Capturing the low-level architectural data available to the VMM such as disk and network I/O accesses, page faults, translation look-aside buffer (TLB) flushes, and control register updates.
  - *Feature Construction* - Using statistical techniques to transform the raw data into *features*, which are used by the regression algorithms.
- A *back-end*, whose functionality includes:
  - *Regression* - Applying multiple linear least-squares regression and the Least Absolute Shrinkage and Selection Operator (Lasso) [6] algorithm to the features generated from a workload in order to characterize and build a model of the workload's behavior.

A high-level overview of our VMM-based workload characterization design is presented in Fig. 1. The front-end extracts the VMM-level events and constructs the features. These features are passed on to the back-end where regression algorithms are employed to build the workload characterization model.

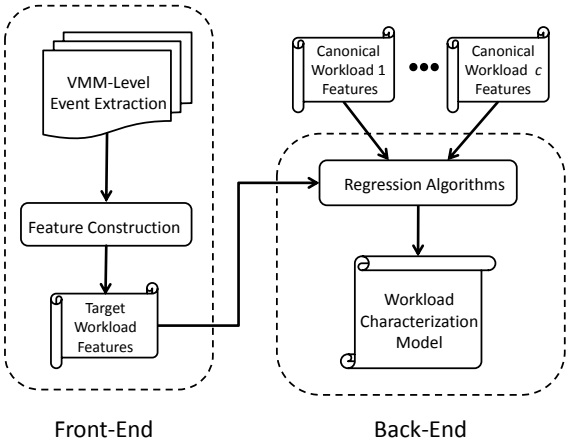


Fig. 1: High-level design of our VMM-based workload characterization tool

In this paper, we provide a general framework for modeling the behavior of workloads executing in virtualized environments that can be applied to a range of domains including intrusion detection for VM security, workload scheduling, and VM health monitoring. In this framework, VMM-level events produced during the workload's execution are captured and processed to generate features containing information about the workload behavior. These features can be further processed in the back-end to produce different outputs, depending on the application of the framework. In the current work, we apply this approach to workload characterization. The input provided to the back-end consists of the features of a set of canonical workloads and a target workload which we would like to profile. As the output, the back-end produces a qualitative model of the workload behavior in terms of a mix of the canonical workload set.

The remainder of the paper is organized as follows. In section II, we survey the literature of related work in the area of virtualization-based workload characterization and the application of regression techniques to the task of performance analysis. In section III, we describe the front-end of our VMM-based workload characterization tool, including the information we are able to extract from the VMM and how it is used to build features. In section IV, we review the approach taken by our back-end to *learn* the behavior of the target workload and characterize it in terms of the canonical workload set. In section V, we describe our experimental setup and present the results of our workload characterization tool on several target workloads. Finally, we conclude the paper and present directions for future work in section VI.

II. Related Work

Prior work on virtualization-based workload characterization has focused on disk I/O workload characterization. Ahmad [7] uses online histograms to analyze the performance of disk workloads running in virtual machines on VMware's

ESX Server. His method is used to characterize several benchmarks, including Filebench – a model-based workload generator for file systems, Open Source Development Labs (OSDL) Database Test 2 – an online transaction processing (OLTP) transactional performance test based on the Transaction Processing Performance Council Benchmark C (TPC-C) specification, and a workload that performs large file copy in virtual machines.

Along the same lines, Gulati, Kumar, and Ahmad [8] perform storage workload characterization and consolidation for enterprise applications using VMware's ESX Server. Their benchmark workloads include Swingbench, DVDStore, Microsoft Exchange, and a TPC-C-like workload running on top of an Oracle database. They characterize workloads by running them inside virtual machines and observing the I/O patterns from the hypervisor using a utility called vscsiStats which generates histograms for various I/O parameters.

Regression algorithms have been previously applied to characterize the performance of workloads. In [9], Ould-Ahmed-Vall, Woodler, Yount, and Doshi present a comparison of several regression algorithms for computer architecture performance analysis of software applications. They compare the following algorithms: multiple linear regression, artificial neural networks, locally weighted linear regression, M5' model trees, and support vector machines (SVM). They collect a series of micro-architectural events, including those related to execution time, instruction mix, branches, memory subsystem, and translation look-aside buffers (TLB). Their data is collected on a subset of the Standard Performance Evaluation Corporation (SPEC) CPU2006 workloads. As part of their results, they present the prediction quality and usefulness of the output from the algorithms, with multiple linear regression and model trees producing easily interpretable results.

In [10], Mousa, Doshi, and Ould-Ahmed-Vall apply multiple linear regression and model tree analysis to characterize performance in virtualized execution. They break up a workload execution into fixed-sized intervals based on the number of retired instructions. Within each interval, they gather counts of events derived from three primary sources: hardware performance monitoring counters (PMC), guest kernels, and the hypervisor (VMM). Using linear regression, the cycles per instruction (CPI) of the execution is decomposed into a linear combination of the architectural and virtualization events they collect. In [11], the authors present a low overhead profiling tool that automates the collection of hardware and software events spanning the vertical execution stack.

In our work, we break up a workload's execution into fixed-size intervals based on (virtual) execution time. The events that we collect are those visible to the VMM. Working completely at the VMM layer offers several advantages. By harnessing the ability of the VMM to isolate and manage several virtual machines, it is possible to provide workload characterization at a common level across all VMs. In addition, being situated within the VMM provides ease of deployment as the tool is not tied to a specific OS and can be deployed transparently below different operating systems.

In their paper, Mousa, Doshi, and Ould-Ahmed-Vall note that due to the dynamic behaviors introduced by virtualization, performance characterization demands high resolution yet minimally intrusive instrumentation. Another benefit of working strictly at the VMM layer relates to the low overhead[1] introduced by our tool as it only requires instrumenting the VMM.

As part of any workload characterization, it may be possible to identify *phases* of execution within a program. In characterizing the behavior of a workload over time, changes in the regression coefficients produced by our workload characterization tool can provide insight into the time-varying behavior of the workload.

Next, we present the details of our VMM-based workload characterization tool.

III. VMM-Based Tool Front-End

The front-end of the VMM-based workload characterization tool has the responsibility of extracting low-level architectural events and subsequently transforming them into features used as input to the regression algorithms. The next section describes the VMM-level events that are captured.

A. VMM-Level Event Extraction

We use the term *events* to describe the raw data and information extracted from the VMM during execution. The information we can extract from different VMMs differs depending on the specific VMM implementation; this can affect the effectiveness and accuracy of the workload characterization tool. The success of a VMM-based tool hinges on its ability to take the extracted events and piece together an accurate picture of the system behavior at the application level.

In our work, we target similar VMMs (in terms of performance, target architecture, etc.) such as VMware Workstation, VirtualBox, ESX Server, and Xen. A subset of events can be found in all of them. These events are *architectural events* that the VMM must intercept to guarantee correct execution. They include execution of privileged instructions (e.g., updates of control registers), access to shared resources (memory), and I/O (disk, network, devices). We rely on this common set of events to create a robust VMM-based workload characterization tool.

The VMM lies below the guest OS layer and provides the illusion that it is running on a real machine. Hence, every time the OS needs to interact with the hardware, the VMM must intervene. It is through this intervention that we are able to collect events. We instrument the VMM to log the occurrence of an event, as well as any available information relevant to the event. For example, when a disk I/O event occurs, the VMM intercepts the request and we record the disk sector accessed, the number of bytes accessed, and read/write status.

[1] An analysis of the current execution overhead of the VMM instrumentation shows that, in terms of wall clock time, the event extraction currently results in about a 10% performance degradation. In other words, adding event extraction to the VMM results in approximately 10% longer execution time.

The data available to the VMM dictates the types of events that we are able to monitor. The richness of the events, in terms of the wealth of information they provide to the VMM, is critical to the success of a VMM-based workload characterization. Consequently, a vast array of events must be collected in order to reconstruct a more accurate picture of the workload activity at the application level. Events can be categorized into two main types:

1) Virtual or VM events - architectural-level and system events related to the virtualized guest OS executing inside the VM. For example, a guest modifying control registers, flushing the TLB, or writing to the disk.
2) VMM events - these events are extracted from the VMM and relate to the state of the VMM itself. For example, VirtualBox has two internal modes: one to execute the guest OS directly on the CPU without intervention (user mode instructions) and another to intercept, instrument, or emulate supervisor mode instructions.

B. Feature Construction

We use the term *features* to describe the information and input format provided to the back-end. The back-end is sent processed information that can be the result of filtering, aggregating, or correlating events. Features do not contain all the information extracted from the events, but incorporate patterns that are effective in identifying relevant execution behaviors.

The space of possible features is very large; there are many ways to process raw events to create features. All of the features we use in this work are constructed directly from the event stream in the following way: first, we divide the event stream into consecutive segments of equal (virtual) time. We apply statistical methods on the events in each segment to produce a vector of feature-values. Finally, we send a *window*, containing a vector of all feature-values for a time segment, to our back-end. For a detailed description of the events extracted and the features that we construct, please refer to [12].

Next, we describe the steps performed in the back-end.

IV. VMM-Based Tool Back-End

At the back-end, the features from the canonical workload set[2] and the target workload are passed to the regression algorithms in order to produce a model of the behavior of the target workload. The resulting model characterizes the target workload in terms of a linear combination of the canonical workloads. The regression algorithms applied in this paper are multiple linear least-squares regression and the Lasso algorithm. The details of these algorithms are provided below.

A. Multiple Linear Least-Squares Regression

Multiple linear least-squares regression [13] can be used to describe a variable in terms of a linear combination of a set of independent variables. We use this notion to describe a target workload as a linear combination of the canonical workloads. We can represent a target workload and each canonical workload as a vector of its different feature-values. In the following regression formula, we use y_i to denote the i^{th} feature of a target workload, x_{ij} for the i^{th} feature of the j^{th} canonical workload, b_j for the coefficient of the j^{th} canonical workload, and e_j as the j^{th} error term.

$$y_i = b_0 + \sum_{j=1}^{c} b_j x_{ij} + e_i \qquad \text{for} \quad i = 1, \ldots, d \tag{1}$$

where c is the number of canonical workloads, b_0 is an optional constant parameter, and d is the number of features. The solution to the linear regression is the set of coefficients $b = (b_1, b_1, ...b_c)$ that minimize the sum of the squared errors. These coefficients comprise the workload characterization model for the target workload.

B. Lasso

We also run experiments on a variant of linear regression that can automatically select the important input variables as well. It performs variable (in our case canonical workload) selection by pushing the coefficients of irrelevant workloads toward zero, giving us a sparse solution (i.e., only a few of the canonical workloads have nonzero coefficients). In particular, it minimizes the least squares error criterion (criterion used in ordinary regression) subject to the constraint that the sum of the absolute value of the coefficients (norm 1) is as small as possible. This algorithm is named Lasso [6] for *Least Absolute Shrinkage and Selection Operator*. The Lasso solution expects many coefficients to be close to zero, and a small subset to be nonzero [14]. Lasso solves the following optimization problem [15]:

$$\min_{\mathbf{b}} \frac{1}{2} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{c} b_j x_{ij} \right)^2 \tag{2}$$

subject to:

$$\sum_{j=1}^{c} |b_j| \leq t \tag{3}$$

where t is a pre-defined threshold. Equivalently, the solution to (2) also minimizes the *Lagrange* version of the problem:

$$f(\mathbf{b}) = \frac{1}{2} \sum_{i=1}^{n} \left(y_i - \sum_{j=1}^{c} b_j x_{ij} \right)^2 + \lambda \sum_{j=1}^{c} |b_j| \tag{4}$$

where $\lambda \geq 0$.

There is a one-to-one correspondence between λ and the threshold t. If $\mathbf{b}(\lambda)$ minimizes (4), then it also solves (2) with $t = \sum_{j=1}^{c} |b_j(\lambda)|$.

The linear regression and Lasso results can be produced for each time window of the target workload, as well as an average-case window. We present both these results in section V-D.

Next, we describe our experimental setup and workload characterization results.

[2]The features for the canonical workloads are created by passing through the front-end of the framework.

TABLE I: Summary of the canonical workload set

Canonical Workload	Description
CPU	A tight loop containing integer arithmetic operations
Memory	A loop that constantly allocates and touches memory without freeing it
Disk Read	Runs 20 threads of IOzone [18], each performing 4 KB reads from a 100 MB file (iozone -i 1 -r 4 -s 100M -t 20)
Disk Write	Runs 20 threads of IOzone, each performing 4 KB writes to a 100 MB file (iozone -i 0 -r 4 -s 100M -t 20)
Network Receive	Runs 4 threads, each constantly reading 60 KB packets over UDP from external senders
Network Transmit	Runs 4 threads, each constantly sending 60 KB packets over UDP to external receivers

V. Evaluation

In this section, we describe the details of our experiments and present the results of applying regression to data extracted from the VMM layer, to produce workload characterization models.

A. Experimental Setup

For our virtual machine monitor, we use VirtualBox, a virtualization environment wherein the VMM runs on a commodity operating system. We use the open source edition version 2.2, currently developed by Sun Microsystems as part of its Sun xVM virtualization platform. All the experiments are executed on a Dell XPS710 machine equipped with an Intel Core2 processor (2 cores) running at 1.86 GHz with 4 GB of RAM. The host operating system is Fedora 9. In the virtual machine, we installed version 9.10 of the Ubuntu OS as the guest operating system. For the Apache and MySQL workloads, the server is executed within the virtual machine, but the client is executed on a separate (non-virtualized) Dell machine with an Intel Pentium III processor running at 866 MHz with 256 MB of RAM and Fedora Core 6 as its operating system.

B. Canonical Workloads

As described in section I, a generic workload can be comprised of the following main components: CPU activity, memory (RAM) activity, disk I/O (read and write), and network I/O (transmit and receive) [4], [16]. As such, we generated a series of workloads targeted to each of these components, which we refer to as the *canonical* workload set. These workloads are based on the benchmarks provided in the Isolation Benchmark Suite [17]. A summary of each workload in the canonical set is provided in Table I.

Once we characterized our canonical workloads, each stressing a different aspect of a computer system, we selected a set of application workloads whose behavior we would like to characterize. These workloads are described next.

C. Application Workloads

To create a diverse set of evaluation workloads, we use the following set of applications: MPrime [19] for Mersenne prime numbers, ApacheBench (*ab*) for Apache web servers, a TPC-C-based online transaction processing benchmark for MySQL databases [20], and mcf from the SPEC CPU2006 benchmark suite. For MPrime, we run one thread performing small fast Fourier transforms (FFTs), wherein the data fits in the L2 cache and the floating point unit (FPU) is heavily stress-tested. For the Apache workload, we installed version 2.2.15 of the Apache web server on the virtual machine. The web page requests were performed from a remote client desktop and for the ab command line, we use the following options: -c 10 (for 10 simultaneous requests) -k (to perform multiple requests in a session) and -n 150000 (for 150,000 requests). For the MySQL workload, we simulate one terminal performing database queries and updates to the stock items of ten warehouses. For the input to the mcf benchmark, we use the *reference* input.

The behaviors of these workloads exhibit a range of the activity that can be represented by a mix of the canonical workloads. For the MPrime workload, we execute torture test number 1, which is predominantly CPU-intensive. We expect the Apache workload to show strong network activity. For the MySQL workload, the MySQL server is run on the virtual machine and the MySQL client is executed on a separate, remote machine. The two machines communicate over an Internet connection. Therefore, in addition to the expected disk read and disk write activity that is typical with processing queries that access database tables, there should also be network activity between the client and servers machines. The mcf benchmark is a pointer-intensive integer benchmark known for its dominant memory behavior [21].

We use regression algorithms to decompose the activity of the target workloads into a linear combination of the activity of the canonical workloads. The results are presented in the following section.

D. Results

Each of the canonical and target workloads are executed for one hour and the VMM-level events are extracted. These events are then processed to generate features for time windows of length 500 interrupt timer events[3] (approximately 2 seconds long). For each target workload, we pass the features of the workload along with the features of all of the canonical workloads to the regression algorithms to produce a model of the workload behavior. The linear least-squares regression results are presented in figures 2 through 5.[4]

The x-axis corresponds to the window index and the y-axis represents the regression coefficients. For each time window, the linear least-squares regression coefficients are produced for each of the canonical workloads. We also use an optional constant parameter, although from the figures we see that its value is close to zero. A large non-zero value

[3]We also experimented with other time quantums and found the results somewhat insensitive to this parameter.

[4]For the purpose of clarity, we zoomed into a portion of the hour-long run, though the pattern seen in the figures are fairly consistent throughout the execution.

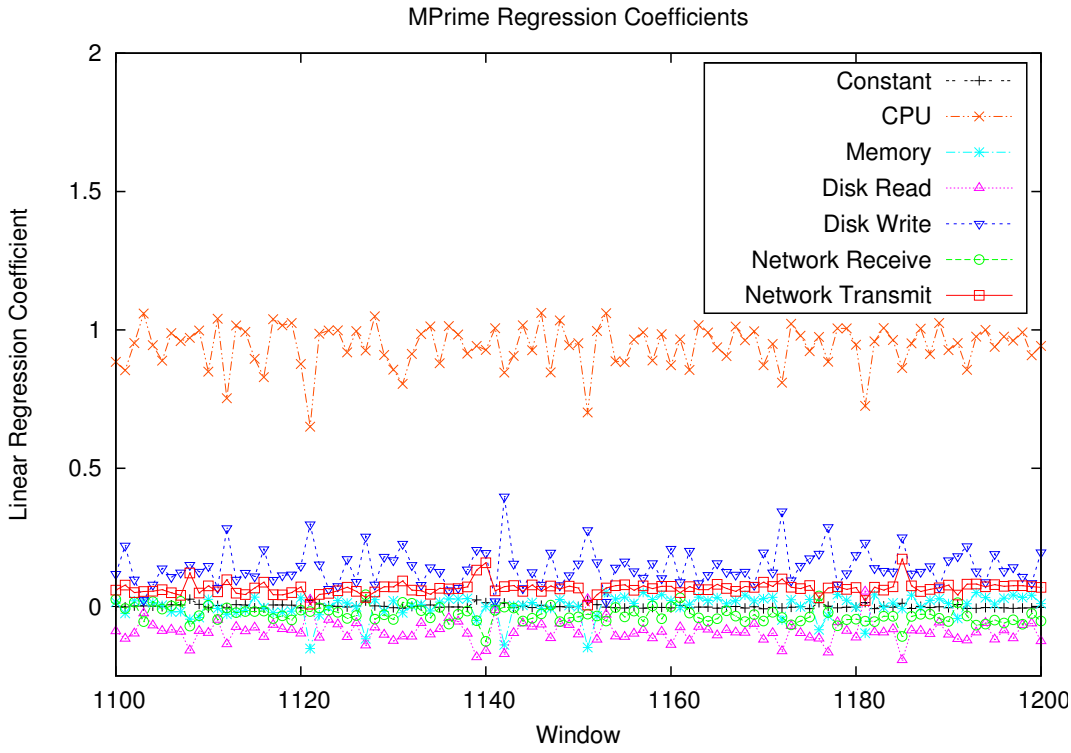


Fig. 2: Linear least-squares regression results for MPrime

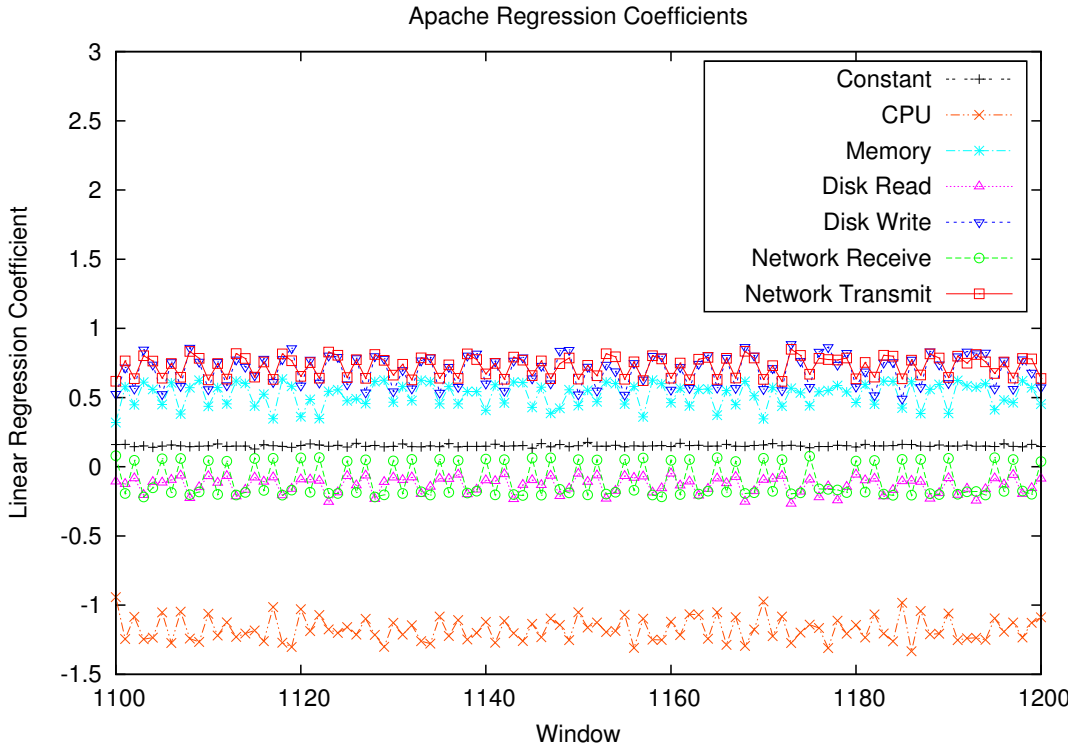


Fig. 3: Linear least-squares regression results for Apache

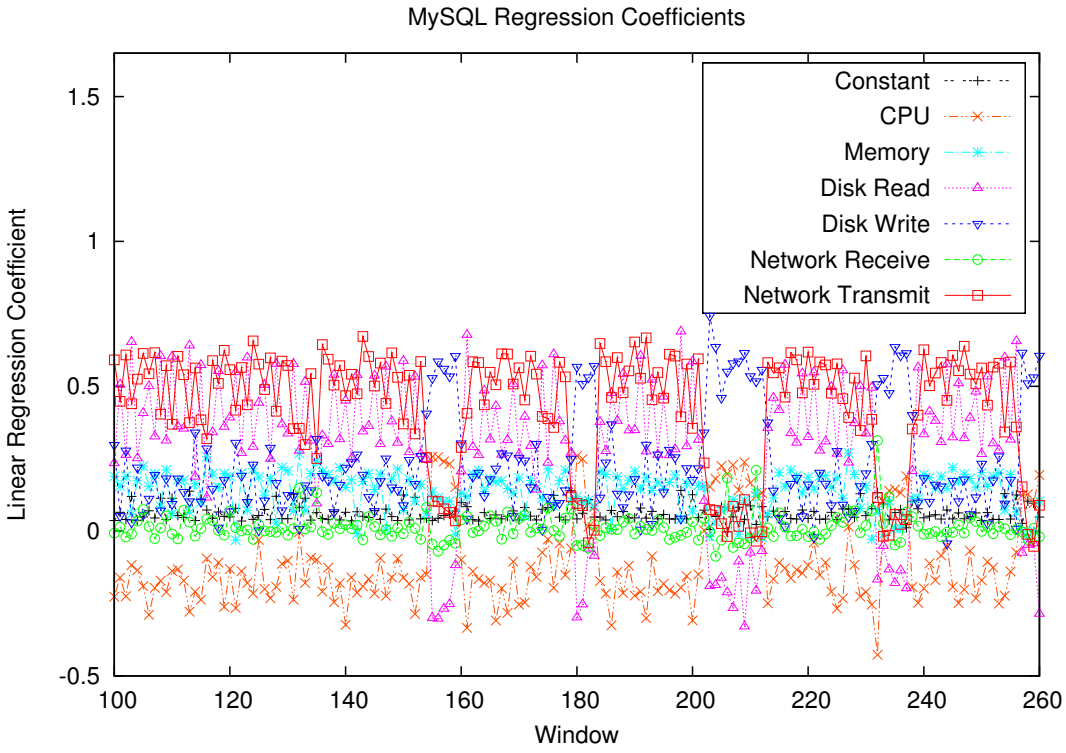


Fig. 4: Linear least-squares regression results for MySQL

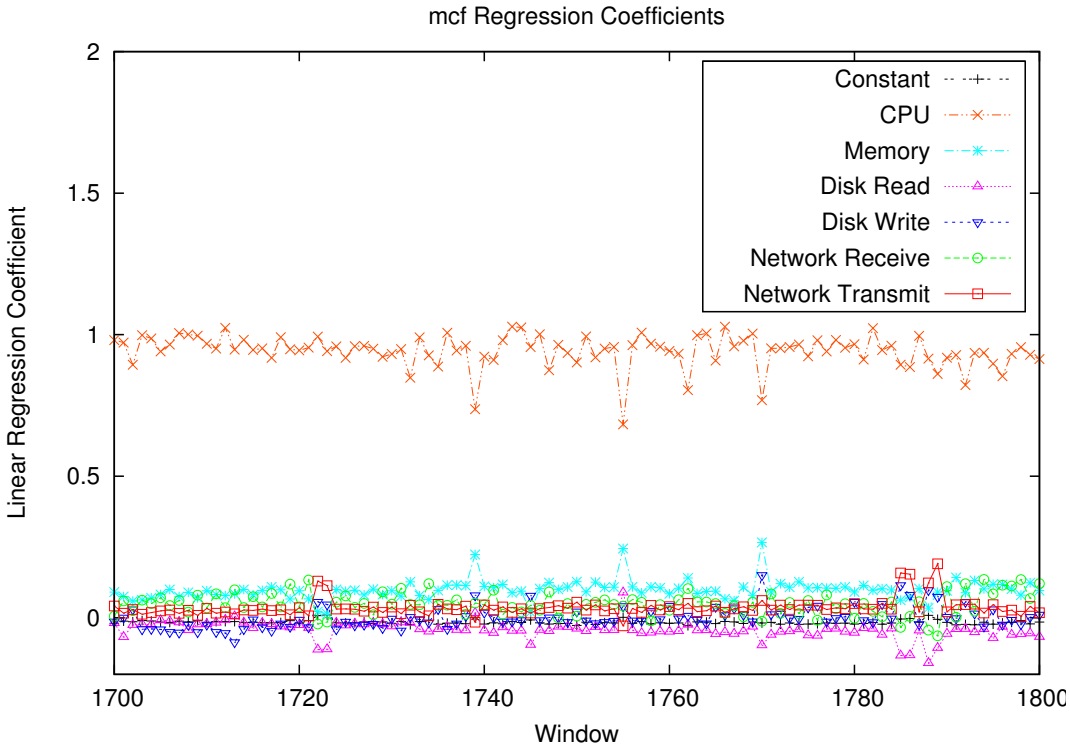


Fig. 5: Linear least-squares regression results for mcf

TABLE II: Linear regression coefficients for an average window of the target workloads

Canonical Workloads \ Target Workloads	MPrime	Apache	MySQL	mcf	mcf using h2
Constant	0.00	0.15	0.07	-0.01	0.00
CPU	0.93	-1.17	-0.07	0.94	0.48
Memory	0.00	0.53	0.14	0.10	0.63
Disk Read	-0.08	-0.14	0.15	-0.04	-0.02
Disk Write	0.14	0.71	0.34	0.00	0.08
Network Receive	-0.03	-0.11	0.00	0.05	0.06
Network Transmit	0.06	0.73	0.32	0.03	-0.01

assigned to the constant parameter would indicate that the workloads represented in the canonical set, by themselves, are not sufficient to characterize the target workload.

To summarize the results, we also find the regression coefficients for the *average window*[5] of each target workload. These results are presented in Table II. For the MPrime workload, the CPU-intensive workload is assigned the largest coefficient, confirming the strong CPU activity of MPrime. For Apache, the results show that the network transmit, disk write, and memory-intensive canonical workloads are assigned large positive coefficients and the CPU-intensive workload is assigned a large negative coefficient. This indicates that the Apache workload exhibits activity similar to that of the network transmit, disk write, memory, and CPU-intensive workloads. Based on an Apache web server's main task of providing web pages to a client machine over a network connection, we expect to see network transmit activity. Using the Linux I/O performance monitoring utility *iostat*, we were able to confirm that the Apache workload also exhibits disk write activity. The Linux *top* utility shows that due to simultaneous web page requests, Apache displays stronger CPU and memory activity than that of a single web page request.

For the MySQL workload, we see an interesting alternating behavior where there is strong network transmit activity in addition to disk activity and then there are short phases where the network transmit activity is low and the disk activity is predominant. Further inspection of the data reveals that during the short phases where the network transmit-intensive canonical workload is assigned a low regression coefficient, the number of network transmit events are indeed low and during phases where it is assigned a large coefficient, the number of network transmit events are more than an order of magnitude larger.[6] It seems that during these short phases, the workload is more focused on performing database queries and updates, and during the longer phases, the results of the queries are transferred back to the MySQL client machine. This shows that our VMM-based workload characterization approach is able to capture these phase changes. Overall, for the MySQL workload we see strong correlation to the disk write, network transmit, disk read, and memory-intensive workloads. This validates our expectations of seeing disk activity due to database table queries and updates, as well as network activity due to the MySQL server (situated within the VM) sending the results of queries back to the client machine over a network connection.

The SPEC CPU2006 mcf benchmark exhibits strong CPU activity with several intervals of activity which resemble the activity of our memory-intensive canonical workload. Since we expect to see a more prevalent memory-intensive behavior, we decided to take a further look at the raw VMM-level data produced for the mcf benchmark and our memory-intensive canonical workload. This analysis revealed that the memory-intensive workload produced many page faults and disk writes, whereas mcf does not exhibit this kind of behavior. This is due to the fact that the memory-intensive canonical workload basically allocates memory in a loop and once the memory is exhausted, pages are swapped back to disk. Mcf shows a different type of memory-intensive behavior, wherein the memory blocks it accesses fit into main memory, resulting in few page faults and thus does not need transfer data to and from the disk.

To validate our hypothesis that mcf exhibits a different type of memory-intensive behavior than our current memory-intensive canonical workload, we decided to replace the canonical workload with another memory-intensive workload that does not allocate so much memory that it requires constant swapping back to disk. We selected the *h2* benchmark in the DaCapo benchmark suite [22] to produce memory-intensive activity, yet without excessive page fault activity. It is a JDBCbench-like in-memory benchmark, executing a number of transactions against a model of a banking application. Fig. 6 and the last column of Table II show the linear regression results produced by replacing the memory-intensive canonical workload with the h2 benchmark. From this figure, we see that a large positive coefficient is assigned to the h2 benchmark. This confirms the similarity of the memory activity between mcf and h2. Using the Linux top utility, we have seen mcf exhibit strong CPU activity as well, and this is validated by the large regression coefficient assigned to the CPU-intensive canonical workload. One thing we must note is that the h2 benchmark is not a *pure* memory-intensive benchmark. In addition to memory activity, it also displays CPU and disk activity. In future work, we will utilize a purely memory-intensive canonical workload that displays behavior typical of most memory-intensive applications.

In addition to multiple linear regression, we also applied the Lasso algorithm to the average window of our target workloads and found the different coefficients assigned to the canonical workloads, using various values of λ. These results were calculated using the GLMNet [14] Matlab source files and are presented in Fig. 7. As the value of λ increases, the coefficient of less relevant workloads are forced to zero while more relevant workloads are assigned non-zero coefficients.

The Lasso results for MPrime also confirm the strong CPU-activity that was shown in the multiple linear regression

[5]The average window is found by taking the average of all the windows during the hour-long execution.

[6]The number of network transmit events go from less than 50 events per second to about 1500 events per second.

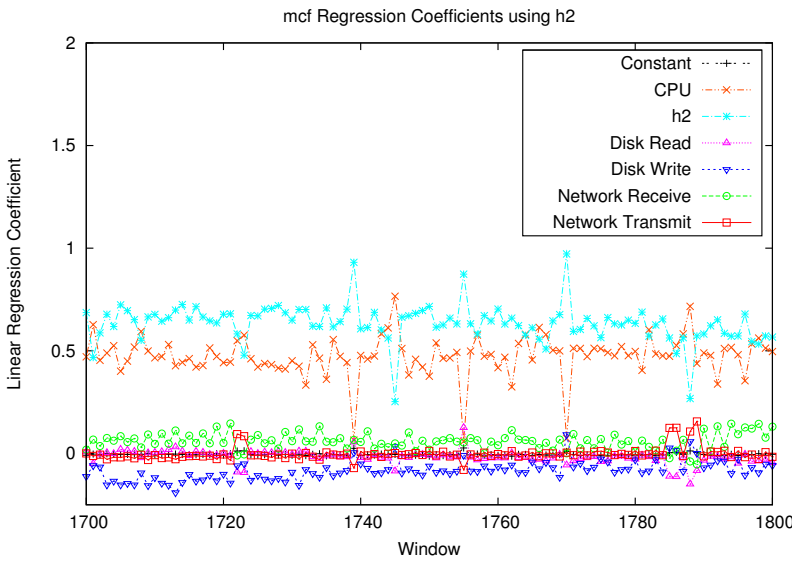
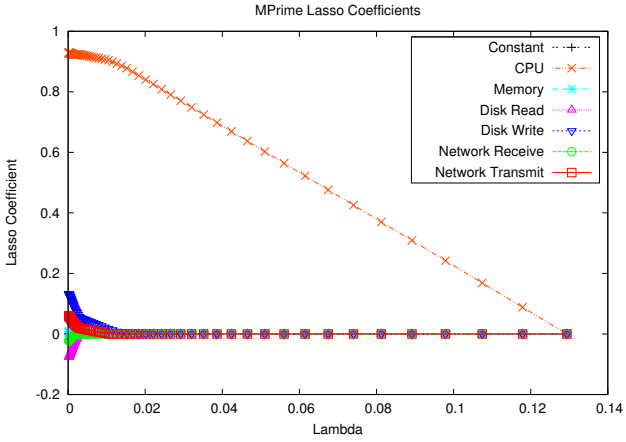
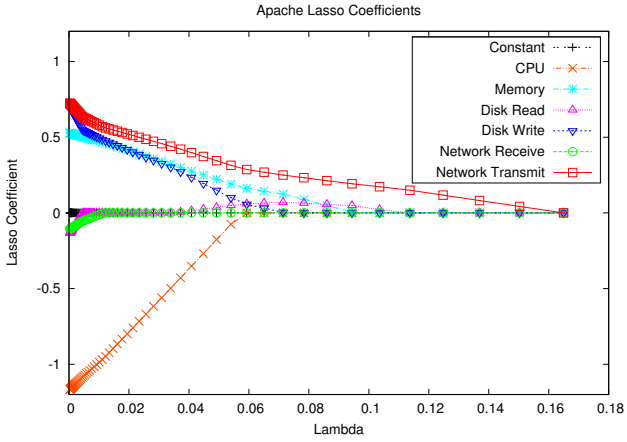


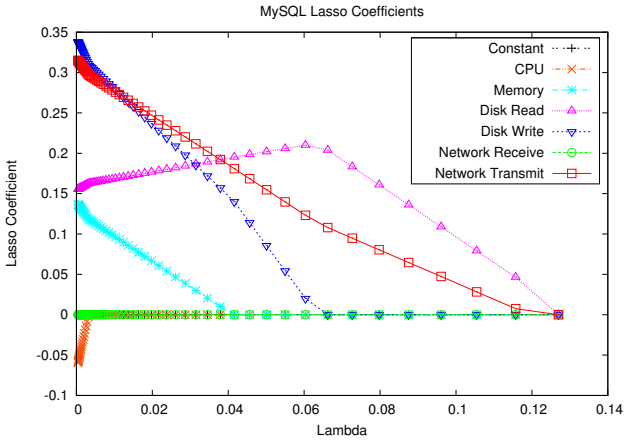
Fig. 6: Linear least-squares regression results for mcf with h2 as the memory-intensive workload



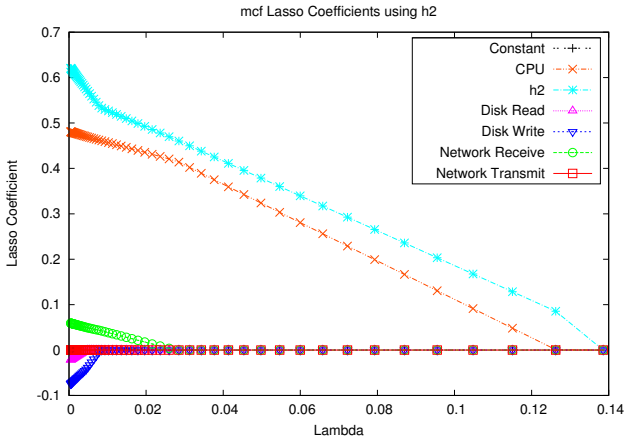
(a) Results for MPrime



(b) Results for Apache



(c) Results for MySQL



(d) Results for mcf

Fig. 7: Lasso regression results for the target workloads

results. For the Apache workload, the canonical workload that retains a non-zero coefficient the longest is the network transmit-intensive workload. Since the Apache ab benchmark services web page requests, transmitting them across a network connection, we expect its behavior to display the most similarity with the network transmit canonical workload.

For MySQL, the most prevalent canonical workloads are the disk read, network transmit, disk write, and memory-intensive workloads. Again, this makes logical sense since MySQL server's main tasks include updating database tables and responding to queries over a network connection. The Lasso results confirm our expectation of MySQL's behavior. For mcf, we produce the Lasso results with the h2 benchmark playing the role of the memory-intensive workload. As a result, h2 is the most prevalent workload, followed by the CPU-intensive workload. This also validates our hypothesis that mcf exhibits memory-intensive behavior similar to that of the h2 benchmark.

VI. Conclusions and Future Work

In this work, we present a general framework for modeling the behavior of virtual execution environment workloads. As a specific application of this framework, we implemented and evaluated a VMM-based workload characterization tool. The open source edition of VirtualBox was used to construct the front-end. We presented the types of information we were able to extract from the VMM and described the procedure used to build features. Regression algorithms were utilized as a powerful technique to bridge the semantic gap between the low-level architectural data and actual program behavior. Our results showed that there is enough information embedded within the VMM-level data to be processed and produce a model of workload behavior in terms of CPU, memory, disk, and network activity.

Our VMM workload characterization tool offers a wealth of future research opportunities. Our front-end can be extended to other VMMs such as Xen or ESX server and its performance can be significantly improved. Additional events can be extracted and used to build new features. Our back-end provides a whole new research dimension. In particular, an area for future work involves studying the model produced from malicious executions and comparing this to the model of normal execution with the goal of providing intrusion detection and enhancing the security of a virtual machine. In the current work, we evaluated target workloads that display a small range of the behaviors present in the canonical workload set. In future work, we can explore the behavior of more realistic workloads that exhibit diverse and time-varying behaviors. This can be extended to perform phase detection and performance optimization.

References

[1] T. Garfinkel and M. Rosenblum, "A virtual machine introspection based architecture for intrusion detection," in *Proceedings of the Network and Distributed Systems Security Symposium*, February 2003, pp. 191–206.

[2] M. Moffie, "Investigating the utility of software semantics for host-based intrusion detection systems," Ph.D. dissertation, Northeastern University, December 2008. [Online]. Available: http://hdl.handle.net/2047/d10018177

[3] M. Alshawabkeh, M. Moffie, F. Azmandian, J. A. Aslam, J. G. Dy, and D. R. Kaeli, "Effective virtual machine monitor intrusion detection using feature selection on highly imbalanced data." in *ICMLA'10*, 2010, pp. 823–827.

[4] H. J. Jeong and S. H. Lee, "A workload generator for database system benchmarks," in *iiWAS*, 2005, pp. 813–822.

[5] Innotek, "Virtualbox." [Online]. Available: http://www.virtualbox.org/

[6] R. Tibshirani, "Regression shrinkage and selection via the lasso," *Journal of the Royal Statistical Society (Series B)*, vol. 58, pp. 267–288, 1996.

[7] I. Ahmad, "Easy and efficient disk I/O workload characterization in VMware ESX server," in *IISWC '07: Proceedings of the 2007 IEEE 10th International Symposium on Workload Characterization*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 149–158.

[8] A. Gulati, C. Kumar, , and I. Ahmad, "Storage workload characterization and consolidation in virtualized environments," in *Workshop on Virtualization Performance: Analysis, Characterization, and Tools (VPACT)*, 2009.

[9] E. Ould-Ahmed-Vall, J. Woodler, C. Yount, and K. Doshi, "On the comparison of regression algorithms for computer architecture performance analysis of software applications." *First Workshop on Statistical and Machine earning approaches to ARchitectures and compilaTion (SMART)*, pp. 116–125, 2007.

[10] H. Mousa, K. Doshi, and E. Ould-Ahmed-Vall, "Characterizing performance in virtualized execution," *2nd Workshop on Statistical and Machine learning approaches to ARchitectures and compilaTion (SMART)*, 2008.

[11] H. Mousa, K. Doshi, T. Sherwood, and E. Ould-Ahmed-Vall, "Vrtprof: Vertical profiling for system virtualization," in *Proceedings of the 2010 43rd Hawaii International Conference on System Sciences*. Washington, DC, USA: IEEE Computer Society, 2010, pp. 1–10.

[12] F. Azmandian, M. Moffie, M. Alshawabkeh, J. Aslam, J. Dy, and D. Kaeli, "Virtual machine monitor-based lightweight intrusion detection," *ACM SIGOPS Operating Systems Review*, 2011, accepted to Operating Systems Review, in press.

[13] R. W. Farebrother, *Linear least squares computations*. New York, NY, USA: Marcel Dekker, Inc., 1988.

[14] J. H. Friedman, T. Hastie, and R. Tibshirani, "Regularization paths for generalized linear models via coordinate descent," *Journal of Statistical Software*, vol. 33, no. 1, pp. 1–22, 2 2010. [Online]. Available: http://www.jstatsoft.org/v33/i01

[15] J. Friedman, T. Hastie, H. Hofling, and R. Tibshirani, "Pathwise coordinate optimization," *The Annals of Applied Statistics*, vol. 1, no. 2, pp. 302–332, 2007.

[16] P. Mehra and B. W. Wah, "Synthetic workload generation for load-balancing experiments," in *Proceedings of the First Symposium on High Performance Distributed Computing*, 1995, pp. 208–217.

[17] J. N. Matthews, W. Hu, M. Hapuarachchi, T. Deshane, D. Dimatos, G. Hamilton, M. McCabe, and J. Owens, "Quantifying the performance isolation properties of virtualization systems," in *ExpCS '07: Proceedings of the 2007 Workshop on Experimental Computer Science*. New York, NY, USA: ACM, 2007.

[18] IOzone filesystem benchmark. [Online]. Available: http://www.iozone.org/

[19] G. Woltman, "Mprime." [Online]. Available: http://www.mersenne.org/freesoft/

[20] Arzuaga, E. and Sridharan, V., "NUCAR OLTP workload implementation." [Online]. Available: http://www.ece.neu.edu/students/earzuaga/research/mysql++-2.0.7.tar.gz

[21] T. K. Prakash and L. Peng, "Performance characterization of SPEC CPU2006 benchmarks on Intel core 2 duo processor," *ISAST Transactions on Computers and Software Engineering*, pp. 36–41.

[22] S. M. Blackburn, R. Garner, C. Hoffman, A. M. Khan, K. S. McKinley, R. Bentzur, A. Diwan, D. Feinberg, D. Frampton, S. Z. Guyer, M. Hirzel, A. Hosking, M. Jump, H. Lee, J. E. B. Moss, A. Phansalkar, D. Stefanović, T. VanDrunen, D. von Dincklage, and B. Wiedermann, "The DaCapo benchmarks: Java benchmarking development and analysis," in *Proceedings of the 21st annual ACM SIGPLAN Conference on Object-Oriented Programing, Systems, Languages, and Applications*. New York, NY, USA: ACM Press, Oct. 2006, pp. 169–190.