

An Artificial Intelligence Framework for Supporting Coarse-Grained Workload Classification in Complex Virtual Environments

Alfredo Cuzzocrea^{*}, Enzo Mumolo, Islam Belmerabet, Abderraouf Hafsaoui

Abstract. We propose *Cloud-based machine learning tools for enhanced Big Data applications*, where the main idea is that of predicting the “*next*” workload occurring against the target Cloud infrastructure via an innovative *ensemble-based approach* that combines the effectiveness of different well-known *classifiers* in order to enhance the whole accuracy of the final classification, which is very relevant at now in the specific context of *Big Data*. The so-called *workload categorization problem* plays a critical role in improving the efficiency and reliability of Cloud-based big data applications. Implementation-wise, our method proposes deploying Cloud entities that participate in the distributed classification approach on top of *virtual machines*, which represent classical “commodity” settings for Cloud-based big data applications. Given a number of known reference workloads, and an unknown workload, in this paper we deal with the problem of finding the reference workload which is most similar to the unknown one. The depicted scenario turns out to be useful in a plethora of modern information system applications. We name this problem as *coarse-grained workload classification*, because, instead of characterizing the unknown workload in terms of finer behaviors, such as CPU, memory, disk, or network intensive patterns, we classify the whole unknown workload as one of the (possible) reference workloads. Reference workloads represent a category of workloads that are relevant in a given applicative environment. In particular, we focus our attention on the classification problem described above in the special case represented by *virtualized environments*. Today, *Virtual Machines* (VMs) have become very popular because they offer important advantages to modern computing environments such as cloud computing or server farms. In virtualization frameworks, workload classification is very useful for accounting, security reasons, or user profiling. Hence, our research makes more sense in such environments, and it turns out to be very useful in a special context like Cloud Computing, which is emerging now. In this respect, our approach consists of running several machine learning-based classifiers of different workload models, and then deriving the best classifier produced by the *Dempster-Shafer Fusion*, in order to magnify the accuracy of the final classification. Experimental assessment and analysis clearly confirm the benefits derived from our classification framework. The running programs which produce unknown workloads to be classified are treated in a similar way. A fundamental aspect of this paper concerns the successful use of data fusion in workload classification. Different types of metrics are in fact fused together using the Dempster-Shafer theory of evidence combination, giving a classification accuracy of slightly less than 80%. The acquisition of data from the running process, the pre-processing algorithms, and the workload classification are described in detail. Various classical algorithms have been used for classification to classify the workloads, and the results are compared.

AMS Subject Classification 2020: 62H30; 68T10

Keywords and Phrases: Virtual machines, Workload, Dempster-Shafer theory, Classification.

***Corresponding Author:** Alfredo Cuzzocrea, Email: alfredo.cuzzocrea@unica.it, ORCID: 0000-0002-7104-6415

Received: 4 July 2023; **Revised:** 5 September 2023; **Accepted:** 6 September 2023; **Available Online:** 7 September 2023; **Published Online:** 7 November 2023.

How to cite: Cuzzocrea A, Mumolo E, Belmerabet I, Hafsaoui A. An artificial intelligence framework for supporting coarse-grained workload classification in complex virtual environments. *Trans. Fuzzy Sets Syst.* 2023; 2(2): 155-183. DOI: <http://doi.org/10.30495/tfss.2023.1990572.1079>

1 Introduction

In this paper, we propose *Cloud-based machine learning tools for enhanced big data applications* (e.g., [34, 7, 21]), where the main idea is that of predicting the “next” workload occurring against the target Cloud infrastructure via an innovative *ensemble-based* (e.g., [47]) *approach* combining the effectiveness of different well-known *classifiers* in order to enhance the whole accuracy of the final classification, which is very relevant at now in the specific context of *Big Data* (e.g., [17]). So-called *workload categorization problem* plays a critical role in improving the efficiency and the reliability of Cloud-based big data applications (e.g., [60, 62]). Implementation-wise, our method proposes deploying Cloud entities that participate to the distributed classification approach on top of *virtual machines* (e.g., [28]), which represent classical “commodity” settings for Cloud-based big data applications (e.g., [37]).

Virtualization technology has become fundamental in modern computing environments such as cloud computing [9, 13, 22, 8] and server farms [57, 23]. By running multiple virtual machines on the same hardware, virtualization allows us to achieve a high utilization of the available hardware resources. Moreover, virtualization brings advantages in security, reliability, scalability, and resource management (e.g., [10, 58, 12]). Resource management in the virtualized context can be performed by *classifying the workload of the virtualized application* (e.g., [66]). As a consequence, workload characterization and prediction have been widely studied during past research efforts (e.g., [14, 3]). More recently, some work has been done on workload characterization in data center environments [26]. On the other hand, workload modeling and prediction in virtualization environments have been addressed in [24, 27, 2], while a virtualized workload balancing approach is described in [29] which uses virtual machine migration, and another approach that focuses on server farms is presented in [61]. From a methodological point of view, *workload classification* is a critical task that integrates the previously-mentioned ones, is performed by collecting suitable metrics during the execution of reference applications, and running a pattern classifier on the collected data, which allows us to discriminate among the different classes. At a base level, the workload can be classified as CPU-intensive or I/O-intensive. In [32], Hu *et al.* perform asymmetric virtual machine scheduling based on this base classification level. At a finer level, the workload can be classified as CPU-intensive, memory-intensive, disk read/write-intensive, and network I/O-intensive. Zhao *et al.* [66] describe a workload classification model based on such a finer classification level. In [65], Zhang *et al.* address the problem of automatically selecting the metrics which provide the best accuracy in the classification task. Also, it has been studied that workloads can be classified by considering memory references as signals, which can be analyzed using spectral parameters (e.g., [53, 42]). Results on instrumented machines and in simulation show that *Hidden Markov Model (HMM) classifiers* [4] can be used to model memory references created and managed by processes under execution.

In our proposed research, the classification phase works as follows. First, in a virtualized environment, we run some programs we take as reference (in this work, we make use of the well-known *SPEC CINT2006 benchmarks* [54]) and, then, from their execution, we extract some features using the APIs of the *Virtual Machine Monitor*. With these so-collected features, we train a model of the workload of each benchmark program according to various and well-understood machine learning algorithms. Unknown programs are executed in the same environment, and their features are fed to models of the reference workloads, in order to find the belief that the unknown workload could be associated with each model. Finally, beliefs obtained by means of different classification algorithms are fused using the *Dempster-Shafer rule of evidence combination* [48] in order to derive a higher-quality classifier (e.g., [6]).

In particular, we discriminate among application workloads. In fact, we run the SPEC2006 benchmarks under a virtualized operating system, and we collect some features through the Virtual Machine Monitor. Using machine learning algorithms, we develop a model for each workload. Unknown workloads are then classified among the different models. The classification among application workloads running in virtualiza-

tion gives interesting potential applications. For example, if the benchmarks are chosen appropriately, it may be determined what the main characteristics of the processes running in the virtual machine are. Another possibility might be to know what are the processes that a given customer typically executes. Other possible applications are in the area of *malware detection* [31]. In this respect, running processes can be monitored to see if their workload is the same or if it changes over time (e.g., [35]). Preliminary experimental assessment and analysis clearly confirm the benefits derived from our classification framework.

The remaining part of this paper is organized as follows. Section 2 considers related work relevant for our research. In Section 3, we highlight the process of workload categorization using common classification algorithms. Section 4 presents the SPEC 2006 Benchmarks used in this work. In Section 5, we provide a description of the virtual environment settings used to elaborate our experiments. Section 6 introduces the aspects on which we based our data analysis (i.e., memory reference and resource demand). In Section 7, we present a detailed description of our methodology along with the used classification algorithms, i.e., Neural Networks, Hidden Markov Models, k-NN, and ARMA. Section 8 demonstrates the fundamentals of the Dempster-Shafer theory of evidence adopted in our approach. Section 9 shows an innovative case study where we describe workload categorization in the context of anomaly detection. In Section 10, we report our extensive experimental assessment and the obtained results. Finally, Section 11 contains conclusions and future work of our research.

2 Related Work

The problem of workload categorization has gained a great deal of attention from researchers, and as a result, several works have appeared in the active literature. In this Section, we will discuss some of the most relevant to our work.

In [36], the authors provide a solution to reduce the risk of incidents and injury in *hazardous work conditions*, especially in the forestry industry, which is one of the most dangerous industries in New Zealand, by proposing a *semantic paradigm for workload classification*. The model takes a collection of *multi-modal physiological measures* as input and categorizes a sequence of workloads (resting, cognitive, and physical workloads). The proposed model was subjected to a series of experimental assessments with participants ranging in age from 22 to 39 based on three different scenarios: (i) relaxing and refraining from any physically or intellectually demanding tasks; (ii) performing a cognitively intense activity; (iii) walking, jogging, and running. The obtained results in these experiments achieved an average accuracy of 89% for resting workload, 76% for cognitive workload, and 97% for physical workload. Finally, the contribution reported in this work, by proposing the model to forecast fatigue in hazardous sectors, opens the doors to a wider research initiative focused on technological applications in hazardous work situations.

[55] presents a *workload categorization-based resource allocation framework* for balancing the load between active physical machines and leveraging their resource capacities. The *CloudSim simulator* is used to run simulation-based experiments using three separate sets of tasks having 10000, 20000, and 30000 tasks. During the experiments, the imbalance in workload among active physical machines and the disparity in resource utilization, specifically CPU and RAM, are observed and measured. According to the simulation results, the proposed framework outperforms similar methods in the literature in terms of balancing the load among active physical machines and using their various resource capabilities.

In [40], they focus the attention on performance testing in new application developments and propose a *performance engineering strategy* that extracts the workload of an existing legacy *Enterprise Resource Planning* (ERP) application with over 1 million users and produces workload for a new version of this application. The proposed method demonstrates that (i) workload for new application testing and architecture validation can be generated from legacy application behavior; (ii) end user organizations have significantly different usage patterns; (iii) high-level operations provide a useful method for analyzing and generating workload

for ERP applications as opposed to low-level page views. The experimental tests of the proposed method performed on a Dutch software firm show that leveraging this approach gives better results in performance engineering.

In [41], the authors investigate and classify *Infrastructure as a Service* (IaaS) cloud workloads into patterns based on their behavioral features as effective characterization of workloads plays a crucial role in driving *Capacity Planning and Performance Management* in IaaS Cloud environments. Various workload metrics, including CPU utilization, memory usage, throughput, and response time, can be leveraged and modeled to understand their interrelationships. Furthermore, different types of behavioral patterns that can be observed within workloads and an outline of statistical techniques to be employed in identifying and determining these patterns are presented in this work. To support their research, they present initial results obtained from the analysis of development workload data collected in a controlled lab environment. These results highlight the potential of the proposed approach in uncovering meaningful workload patterns and highlight the importance of effective workload characterization for efficient Capacity Planning and Performance Management in IaaS Clouds.

[44] introduces the analysis of Cloud workloads and evaluation of the effectiveness of two commonly used prediction techniques, namely *Markov Modeling* and *Bayesian Modeling*, using a dataset comprising 7 hours of Google cluster data. The primary objective is to assess the performance of these methodologies in accurately forecasting user demand. Moreover, a key aspect of this study involves the categorization and characterization of Cloud workloads, which enables the modeling of essential parameters for user demand forecasting. By understanding the patterns and characteristics of workloads, the authors aim to enhance the accuracy of demand prediction, thereby facilitating efficient resource allocation and energy consumption management, they present an optimal solution to minimize idle resources and reduce unnecessary energy consumption while ensuring *Quality of Service* (QoS) maintenance. Through the experimental analysis and assessments, the research provides insights into the effectiveness of different prediction methodologies for Cloud workload forecasting. This research contributes to the development of energy-efficient Cloud environments while maintaining optimal QoS levels.

In [51], the authors manage the *dynamic scalability of resources* in IaaS environments by studying different workloads and classifying them based on their features and limits. Additionally, metrics aligned with QoS requirements are defined and analyzed for each task, enabling the creation of improved application architectures, as efficiently managing these workloads is essential for the *optimal utilization of dynamic natural resources*. This research contributes to enhancing the efficiency of Cloud resource utilization by considering workload as a core capacity. Therefore, by effectively classifying and characterizing workloads, organizations can optimize resource allocation and ensure that QoS demands are met. This research emphasizes the importance of aligning application architecture with workload characteristics and QoS metrics to achieve optimal performance in IaaS Cloud environments.

[49] discusses the problem of *Instruction Set Architecture* (ISA)-independent workload characterization for significant program features related to compute, memory, and control flow by employing a *Just-In-Time* (JIT) compiler that generates ISA-independent instructions. Through a comparative analysis with an x86 trace, they evaluate the impact of different ISAs on the workload characterization results, revealing that certain aspects of the study exhibit significant sensitivity to the ISA employed. This highlights the importance of adopting ISA-independent workload characterization methodologies for designers of specialized architectures. Based on these results, one can notice that specialized architecture designers must utilize ISA-independent workload characterization methodologies to ensure accurate and reliable assessments of program features. By decoupling workload characterization from specific ISAs, designers can effectively optimize specialized architectures for energy efficiency while considering the unique demands and characteristics of the workload, providing insights into the design and development of energy-efficient computing solutions in the industry.

3 Operational Principles

The training and testing phases of the classification algorithm are described in Figure 1. The idea behind training is to use the different execution sequences produced by a program when fed by different inputs to train the workload model of that program. On the other hand, when an unknown execution sequence is given to a workload model, the probability that the workload of the unknown sequence is similar to that of the model is produced.

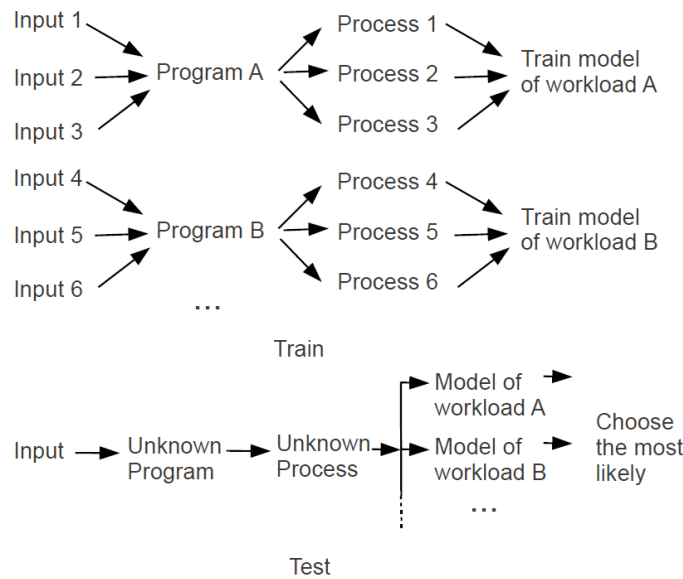


Figure 1: Training and testing of workload models

It is also worth remarking that a number of inputs are given to the benchmarks. In other words, we generate different executions from a given benchmark using different input data. The executions generated from the same benchmark are different because they are obtained with different inputs. Nevertheless, the executions have in common the fact that they come from the same benchmark. The correct classification of the workload of one process means that the classifier is able to understand that different executions come from a single benchmark. Furthermore, we perform the workload classification using four classifiers, namely *Neural Networks* (e.g., [46]), *Hidden Markov Models*, *k-Nearest Neighbors* [1] and *ARMA* [33].

After that, we show that the Dempster-Shafer data fusion algorithm can be successfully used with two different and independent types of metrics. The final classification rate is slightly less than 80% over six benchmarks. In this work, we derived workload models from six benchmarks.

We performed two classification experiments: first, we tested the workload models with the same six benchmarks used to derive the models. However, the input data is different from that used in training, and therefore the processes are always different. Secondly, the other six benchmarks are used for evaluating the similarity with the workload models.

Different classifiers and features can be considered in a data fusion framework to improve classification accuracy, as reported in Figure 2. As it will be shown at the end of the paper, these higher-quality classifiers can be used to find the category of an unknown workload.

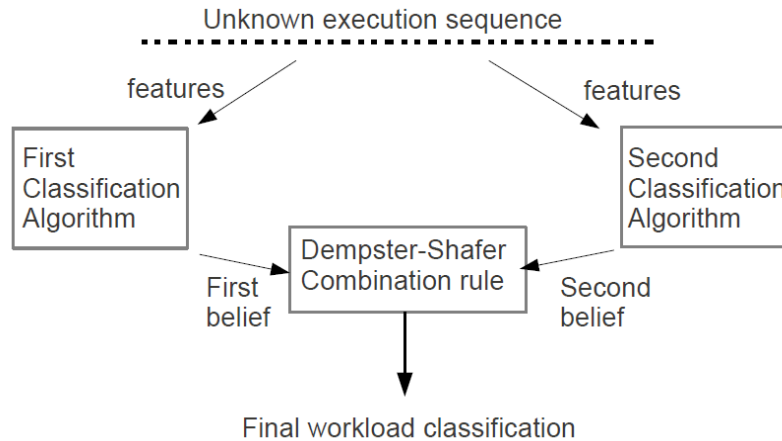


Figure 2: Data fusion of two classifiers

4 SPEC 2006 Benchmarks

CINT2006 [54] is SPEC's CPU-intensive benchmark suite, stressing a system's processor, memory subsystem, and compiler. SPEC designed CPU2006 to provide a comparative measure of compute-intensive performance across the widest practical range of hardware using workloads developed from real user applications. All the benchmarks are provided as source code. The twelve programs included in the benchmark suite can be grouped into the following classes according to their functionality: compiler class, game class, compression class, scientific computing class, and optimization class.

In this work, we derived workload models from six benchmarks, namely *401.bzip2*, *403.gcc*, *458.sjeng*, *471.omnetpp*, *400.perlbench* and *462.libquantum*. In the first experiment, we tested the derived models with the same six benchmarks. It is important to note that the input data is different from that used in training, and therefore the execution sequences are always different. In the second experiment, the other six benchmarks are used to evaluate the similarity between the workload models.

It is worth observing that the used benchmarks represent only a fraction of what applications look like because I/O and memory activity are missing. Thus, the reported results have to be considered as preliminary from a general point of view, being valid only within computer-intensive workloads.

It is important to describe how the input data for the benchmarks are organized in order to make the reported results repeatable. SPEC gives six different inputs for *bzip*, nine for *gcc* and three for *perlbench*. The *sjeng* benchmark has only one input; two other inputs for *sjeng* have been obtained from the first chess positions of *chess.html* downloaded from WWW.DOWNSCRIPTS.COM/CHESS-DATABASE.

Similarly, *omnetpp* has only one input furnished by SPEC; other inputs have been obtained from the first example networks reported in [HTTP://INET.OMNETPP.ORG/DOC/INET/NEDDOC/](http://INET.OMNETPP.ORG/DOC/INET/NEDDOC/).

Finally, *libquantum* was given the following two additional pairs of numbers: (159, 15) and (1413, 17). In this way, all the benchmarks have at least three inputs that are used for training. Additional input for the test has been obtained in a similar way.

5 Virtual Machine Setting

The virtualization infrastructure used in this work is provided by VirtualBox, which is an open source full virtualization Virtual Machine Monitor (VMM) that runs on both Linux and Windows operating systems

running on x86 and x64-based architectures [43]. The useful thing is that VirtualBox offers a rich set of APIs that easily allow to collect metrics on the virtualized process, and the complete set of available APIs is described in [56]. The SDK provided with Virtual Box allows third parties to develop applications that can directly interact with it. It is designed in levels, and at the bottom level, we find the VMM (hypervisor) which is the heart of the virtualization engine that allows for monitoring the performance of virtual machines, providing security, and ensuring the absence of conflicts between virtual machines and the host. Above the hypervisor, there are modules that provide additional functionality, for example, the RDP server (*Remote Desktop Protocol*). Finally, there is the API level, which is implemented above these functional blocks.

VirtualBox comes with a web service that, once running, acts as an HTTP server, accepts SOAP connections [52] (*Simple Object Access Protocol*) and processes them. And the interface of this service is described in a *Web Services Description Language* (WSDL) file [59]. In this way, it is possible to write client programs in any programming language that has provided the tools to process WSDL files, such as Java, C++, NET PHP, Python, and Perl. In addition to Java and Python, the SDK contains many libraries that are ready for use. Internally, the API is implemented using *Component Object Model* (COM) as a reference model. In Windows, it is natural to use Microsoft COM, however, in other hosts, where COM is not present, XPCOM, which is a free implementation of COM, can be used.

Despite the numerous advantages of Web service API, we used the COM method because it allows a lower overhead and thus a higher data rate. We conducted a data exchange experiment, and it turns out that the web service is able, on average, to collect data every 5.96 *ms*, while using COM, data can be collected every 0.49 *ms*. As other interesting features regarding the collection of statistical data about resources usage, the API provides functions for:

- specifying which groups of indicators we are interested in (CPU, RAM, Network, and Disk);
- setting the measuring range (minimum interval of 1 *s*);
- setting the frame size for statistics (Min, Max, and Average);
- making queries on the usage of a single resource.

6 Metrics

As regards metrics, we developed the acquisition system described in the block diagram reported in Figure 3. The acquisition is driven by the host, and all the commands and the acquired data use the COM interface. There is also a web interface to the VirtualBox VMM, but it is much slower, as specified above.

The measured quantities used for workload characterization are of two types, namely memory references and resource demand. Both quantities are gathered using the VirtualBox VMM's API classes. The memory references are the instruction addresses generated by the virtualized process. The VMM's IMachineDebugger API can collect the value of the Program Counter related to instructions every 0.5 *ms*. In Figure 4 (a), we report, as an example, a chunk of memory references collected during the virtualized execution of one SPEC benchmark (gobmk). On the other hand, Figure 4 (b) presents a chunk of memory reference for another SPEC benchmark (perlbench).

The data is collected in a $\langle time\ stamp \rangle \langle address \rangle$ format. Using the IPerformanceCollector API, we collect resource demand features generated by the virtualized process. The resource demand features we acquired are the following:

- the CPU used in user mode;
- the CPU used in system mode;

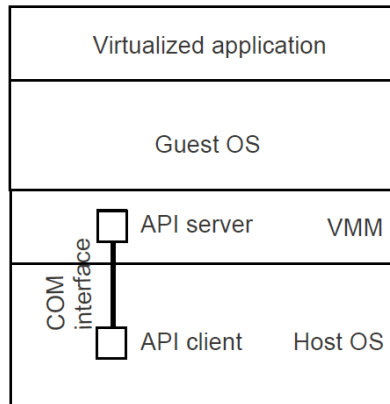


Figure 3: Acquisition system

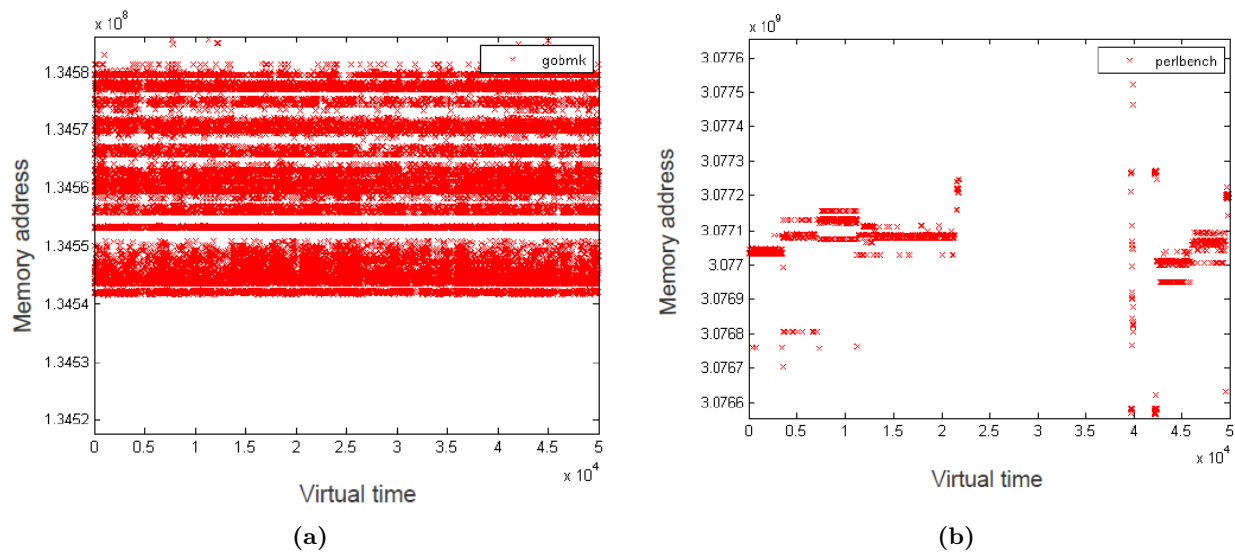


Figure 4: Memory references generated by (a) the *gobmk* benchmark - (b) the *perlbench* benchmark

- memory fragmentation (free memory / total memory).

In Figure 5 (a), we report, as an example, a portion of 400 s of the CPU used when in user mode collected during the execution of the virtualized process. In Figure 5 (b), we report, instead, the amount of free RAM memory available during the execution of the virtualized process.

Each curve is related to the execution of a different process in the virtual machine, and resource demand feature data is also acquired in a time-stamp format as shown in Figure 6, which reports a piece of resource demand metrics acquired during execution.

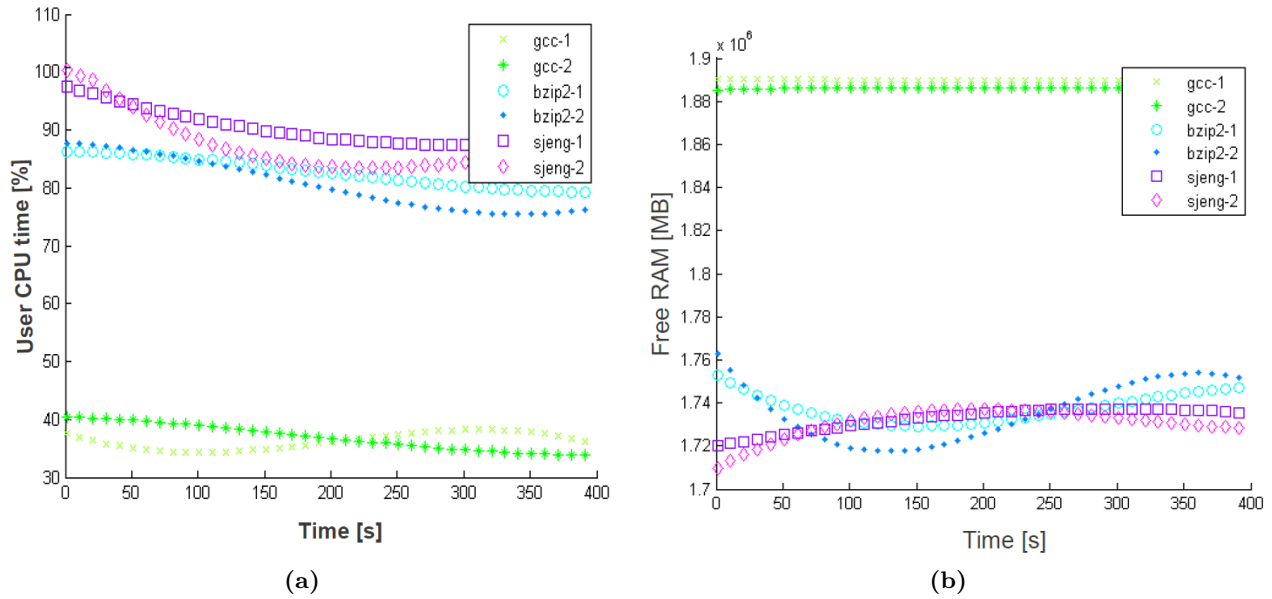


Figure 5: User CPU time feature (a) - free RAM feature (b) for different programs in two different executions

Timestamp	CPU user	CPU system	Free Ram
701893535744118	90.0	10.0	1016784.0
701893637910209	90.0	10.0	1016784.0
701893741889815	90.0	10.0	1016784.0
701893845900845	90.0	10.0	1016784.0
701893949900057	90.0	10.0	1016784.0
701894051790540	90.0	10.0	1016784.0
701894153906958	90.0	10.0	1016784.0
701894257888280	90.0	10.0	1016784.0
701894361898999	90.0	10.0	1016784.0

Figure 6: Resource demand format

7 Data Analysis Methodology

7.1 Pre-Processing Algorithms

It is well known that the initial instructions of a running code are highly non-representative of the steady-state behavior of the program. In fact, the first billion instructions do very little except for file I/O and memory allocation as data structures are set up and populated before getting to the real computation to be performed by the program. In this work, we do not use techniques for discovering program phases such as those described in [50] to find the beginning of the steady-state phase of the programs. Instead, we simply blindly fast forward for 1 billion instructions before starting data analysis.

We consider the memory reference sequence as a one-dimensional signal and the resource demand sequence as a multi-dimensional signal. Similarly to what happens in signal processing, we use a parametric description of the sequences. We remark that events in the process, such as for examples loops or sequential program behaviors, produce important events in the metrics sequences and then in the signal spectrum. For instance, loops introduce peaks in the spectrum, while a sequential address sequence produces a DC spectral component. Moreover, the sequences dynamically change their properties. For these reasons, we used a short-time spectral

description of the memory reference sequence. Thus, the sequences are divided into overlapped analysis frames of a given size, as reported in Figure 7. The frames are further divided into blocks. Hence, for instance, a frame of memory references is represented by a set of blocks. Also, the multi-dimensional sequence of resource demand features is divided into overlapped frames and blocks.

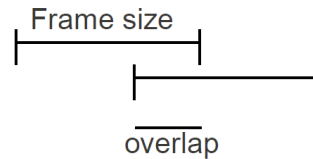


Figure 7: Analysis frames

The next step is to perform a spectral analysis of the blocks. Among the possible spectral-related parameters, we chose the *Discrete Cosine Transform* (DCT) representation. DCT is a well-known signal processing operation with important properties [30]. For example, it is useful for reducing signal redundancy since it places as much energy as possible in as few coefficients as possible (energy compaction). The first DCT coefficients are given as input to the classification algorithm. The effects of retaining the first DCT coefficients are shown in Figure 8. A frame of 1024 memory references is plotted in this Figure. On this frame, we perform a DCT transform; the first sixteen coefficients are used to obtain a 1024-coefficient vector with zero-padding. By inversely transforming this sequence, we obtain the curve plotted in the same Figure 8. It is evident that the effect of retaining the first coefficients is to smooth the peaks of the original sequence while still representing the overall sequence behavior, reflecting the signal redundancy reduction property.

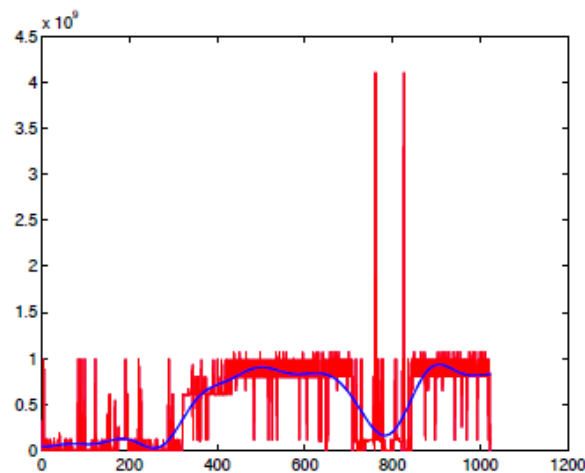


Figure 8: Signal reconstruction by inverse transforming the first DCT coefficients

Concerning the resource demand, since there are three types of features in a frame, the DCT is applied separately to each feature. In every case, we take a small number of DCT coefficients per feature to represent the frame. In this case, a frame is spectrally described by a three-component vector, each formed by a DCT coefficient for a single feature. Eventually, the mono or multi-dimension DCT representations are vector quantized with a 128 entries codebook. The result is that each block of the acquired sequences, both in terms of memory references and resource demand, is described by an integer number.

7.2 Process Selection

The SPEC CPU2006 is formed by two sets of benchmarks: CINT2006 benchmarks, integer benchmarks, and CFP2006, floating point benchmarks. In our experiments, we consider all the CINT2006 benchmarks, formed by twelve programs. Early classification experiments gave the impression that some benchmarks were classified as the same workload. To explore this impression, we performed the following experiment. Using the analysis algorithm of Section 7.1, we train a three-hidden-layer Neural Network for each benchmark, using three different input sets per benchmark. Hence, we have a Neural Network trained for each workload. Then, the vector-quantized parametric sequence obtained from each benchmark in execution was given as input to the Neural Network. The output is very close to one if a given workload is given as input to the Neural Network trained for the same type of workload, and between one and zero for all the others. A distance matrix among all the twelve workloads is obtained by computing $y = 1 - out$, where out is the Neural Network output, and averaging y for the same type of workloads. The size of the distance matrix is clearly twelve by twelve.

By k-means clustering of the distance matrix, we have a reduction to six processes, which confirms our first impression. To get a graphical view of this, in Figure 9, we report the 3D graphical plot, obtained with multidimensional scaling, of the distance matrix. From this Figure, we can see that the distance among *astar*, *h264*, *gobmk*, and *perl* is very small, so they can be represented by only one program. Similarly, the distance between *hmm* and *sjeng* is very small and the distance among *libquantum*, *mcf*, and *xalancbmk* is also very small. In conclusion, the workloads resulting from the clustering reduce to the six benchmarks: *bzip*, *gcc*, *sjecg*, *omnetpp*, *perlbench*, and *libquantum*. The workload classifications reported in the following are performed using these six benchmarks.

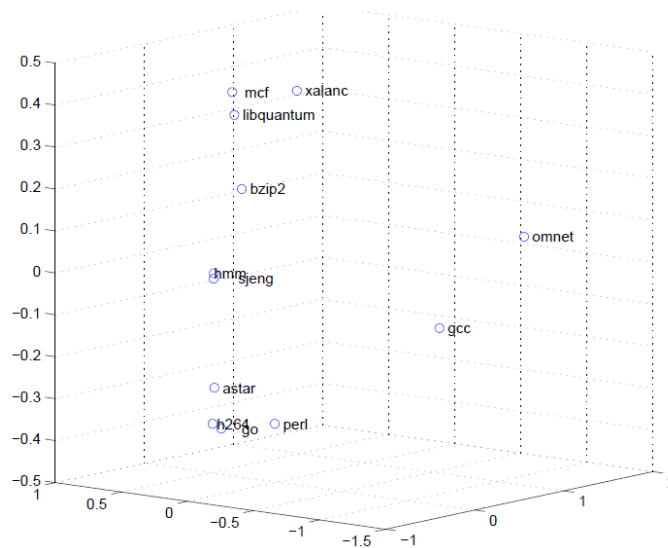


Figure 9: 3D visualization of the distances among programs

7.3 The Input Data to the Benchmarks

Each benchmark has three sets of input data supplied by SPEC in a text file. The sets of input data provided by SPEC are used for training the classifiers. Each benchmark, when run, gives rise to three different processes, one for each set of input data. We can say that each process represents the workload of the benchmark that generated that process. The processes are used to train the classifiers. Each classifier is

trained with the different processes so that you can classify the workload of the benchmark.

For each benchmark, we constructed three other sets of input data, other than the above, for the classification stage. Each process generated during benchmark execution with the new sets of input data is classified by the classifiers trained earlier. This generates a number of classification experiments equal to $3 \cdot N$, where N is the number of benchmarks.

7.4 Sampling Rate

A memory reference value is acquired every 0.5 *ms*. It is important for computational complexity reasons to ascertain how much this sample rate can be reduced. We therefore performed classification experiments with memory reference features at various sampling rates.

For each benchmark, with three different inputs, a Neural Network was trained with the parameters reported in Table 1.

Table 1: Initial analysis parameters

Frame size	50s
Overlap	50%
Number of DCT coeffs	10
Number of blocks per frame	50
Vector quantization	128 levels

In Figure 10 (a), we show the accuracy obtained at different values of the decimation of the original sampling rates. We note that by acquiring the memory references and decimating the original sampling rates by ten, the accuracy drops from 60% to 53%. In view of the fact that this accuracy is improved if the parameters are tuned, and for reducing the algorithm complexity, we acquired the memory references are acquired at a 5 *ms* sampling rate, which corresponds to a decimation factor of ten. The resource demand features acquisition was decimated accordingly.

Starting from the initial analysis parameters, we conducted some experiments of *Neural Network Classification*, using three hidden layers and 50 neurons. First, we obtained the accuracy with different values of frame size. We find that by slightly decreasing the frame duration from 50 *s* to 48 *s* we get a slight accuracy improvement. We also find that the frame overlap we used initially lead to the best accuracy. The next set of experiments is directed to the number of DCT coefficients. We found that the best accuracy is obtained by setting the number of DCT coefficients to sixteen, which is shown in Figure 10 (b).

Other classification experiments indicate that the best number of blocks per frame is forty. Thus, the final optimal analysis parameters are reported in Table 2.

Table 2: Final parameters setup

Frame size	48s
Overlap	50%
Number of DCT coeffs	16
Number of blocks per frame	40
Vector quantization	128 levels

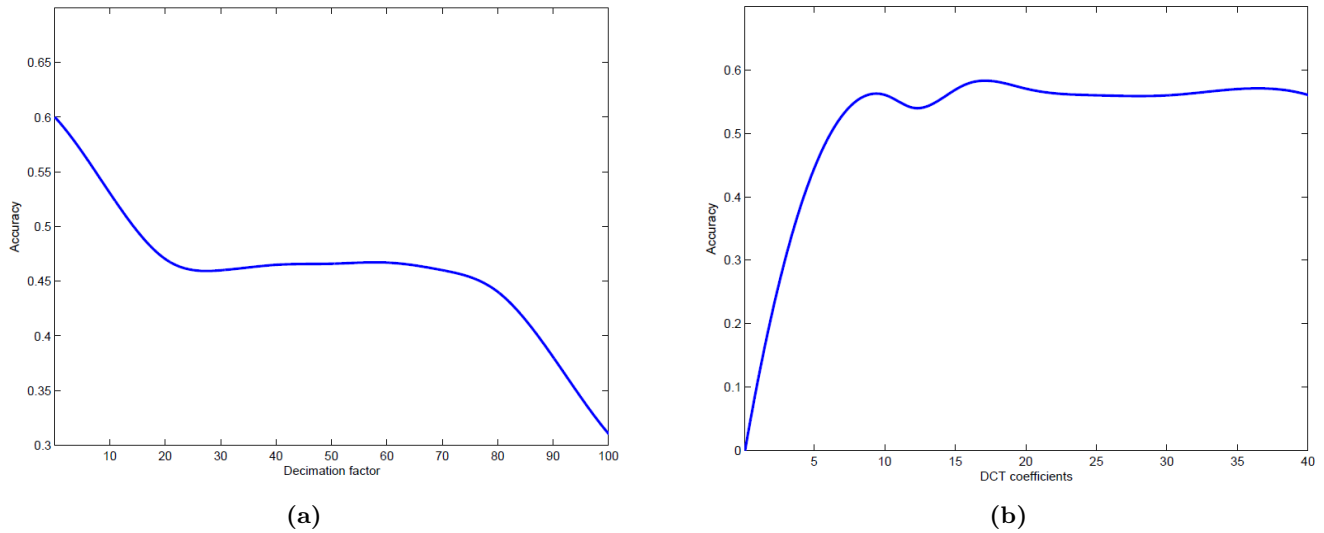


Figure 10: Accuracy versus decimation factor (a) - DCT coefficients (b)

7.5 Classification Algorithms

For the sake of completeness, in this Section we present a brief description of the machine learning algorithms that we used as classifiers.

7.5.1 k-Nearest Neighbor Algorithm

The *k-Nearest Neighbor algorithm* (k-NN) is a simple machine learning algorithm that does not use any underlined model acquired during the training phase, as other machine learning algorithms do. Instead, k-NN is based on the principle that instances within a dataset will generally exist in close proximity to other instances that have similar properties. If the objects are tagged with a classification label, they are classified by a majority vote of their neighbors and are assigned to the class most common amongst their k-nearest neighbors.

k is usually a small odd positive number, and the correct classification of the neighbors is known a priori. The objects can be considered n -dimensional points within an n -dimensional instance space, where each point corresponds to one of the n features describing the objects. The distance or closeness to the neighbors of an unclassified object is determined by using a distance metric (also called the similarity function), for example, the Euclidean distance or the Manhattan distance.

Our k-NN uses the Euclidean distance to represent the closeness to the neighbors of an unclassified object, as the high degree of local sensitivity makes k-NN highly susceptible to noise in the training data. In other words, the value of k strongly influences the performance of the k-NN algorithm.

7.5.2 Neural Network Algorithm

The well-known neural networks are composed of a set of simple processing units which communicate with each other through a large number of weighted connections. In most cases, it is assumed that each neuron makes an additive contribution to the neuron to which it is connected. The total input s_k is simply the weighted sum of the different outputs of the connected neurons plus a noise factor:

$$s_k(t) = \sum_j w_{jk}(t)u_j(t) + \theta_k(t) \quad (1)$$

where positive w_{jk} are said to excite the neuron input, and negative w_{jk} are said to inhibit the neuron.

We need to have rules to determine the effect of the total input on the activation of the neuron. It is well defined a function F_k that, based on the total input $s_k(t)$ and the current activation of the neuron $y_k(t)$ produces the new value of the activation:

$$y_k(t+1) = F_k(y_k(t), s_k(t)) \quad (2)$$

Very often, F_k depends only on the total input at that moment, and then the last Equation can be written as follows:

$$y_k(t+1) = \sum_j w_{jk}(t)y_j(t) + \theta(t) \quad (3)$$

Normally F_k has values in the range $[-1, \dots, +1]$. The most commonly used functions are the *sign function*, the *hyperbolic tangent*, or the *sigmoid function*.

We used a neural network with a Feed-forward topology with three hidden layers, where the flow of information between the input and the output travels one-way to the exit.

- feed-forward networks: here, the flow of information between the input and the output travels one way to the exit. Processing can be extended to many levels of neurons;
- recursive networks: contrary to feed-forward networks, they have feedback connections. The network is bound to evolve towards a stable state in which the functions of activation do not change.

As a classic example of the first type is the perception, while the second type is the *Hopfield network*, a neural network must be configured in such a way that the application of inputs produce the desired output. It is therefore necessary to modify the weight of the connections during a training phase.

7.5.3 Hidden Markov Model Algorithm

Markov models are stochastic interpretations of time series. The basic Markov model is the Markov chain, which is represented by a graph composed by a set of N states, the graph describes the fact that the probability of the next event depends on the previous event. A Markov chain is described by the transition matrix A whose elements are:

$$a_{i,j} = Prob(S_{t+1} = j | S_t = i) \quad (4)$$

and the initial probability vector π_i :

$$\pi_i = Prob(S_1 = i) \quad (5)$$

where:

$$\sum_{i=1}^N \pi_i = 1 \quad (6)$$

In homogeneous Markov chains, the transition probability depends only on the previous state; in such cases, the transition probabilities can be represented by a transition matrix. However, in many cases, Markov models are too simple to describe complex real-life systems and signals.

In the case of Hidden Markov Models (HMMs), the output of each state corresponds to an output probability distribution instead of a deterministic event. That is, if the observations are sequences of discrete symbols chosen from a finite alphabet, then for each state, there is a corresponding discrete probability distribution that describes the stochastic process to be modeled. In HMMs, the state sequence is hidden and can only be observed through another set of observable stochastic processes. Thus, the state sequence is recovered with a suitable algorithm on the basis of optimization criteria. It is important to note that the observation probabilities can be discrete or continuous feature vectors.

7.5.4 ARMA

Considering the memory reference sequence as a time series of data M_t , the ARMA model is a tool for understanding and, perhaps, predicting future values in this series. The model consists of two parts, an *Auto Regressive* (AR) part and a *Moving Average* (MA) part. Thus, the model is referred to as the *ARMA*(p, q) model, where p is the order of the auto-regressive part and q is the order of the moving average part:

$$M_t = c + \epsilon_t + \sum_{i=1}^p \varphi_i M_{t-i} + \sum_{i=1}^q \psi_i \epsilon_{t-i} \quad (7)$$

where φ_i and ψ_i are the parameters of the model, ϵ_t is white noise, and c is a constant. Classification with ARMA model is performed using the generalized linear model.

8 The Dempster-Shafer Fusion

The goal of the Dempster-Shafer (DS) theory of evidence [48], is to combine different measures of evidence. At the base of the theory is a finite set of possible hypotheses, say $\theta = \{\theta_1, \dots, \theta_K\}$.

8.1 Basic Belief Assignment

The Basic Belief Assignment (BBA) can be viewed as a generalization of a probability density function. More precisely, a basic belief assignment m is a function that assigns a value in $[0, 1]$ to every subset \mathcal{A} of θ that satisfies the following conditions:

$$\sum_{\mathcal{A} \subseteq \theta} m(\mathcal{A}) = 1, \quad m(\emptyset) = 0 \quad (8)$$

It is worth noting that $m(\mathcal{A})$ is the belief that supports the subset \mathcal{A} of θ , not the elements of \mathcal{A} . This reflects some ignorance because it means that we can assign belief only to subsets of θ , not to the individual hypothesis.

8.2 Belief Function

The belief function, $bel(\cdot)$, associated with the Basic belief assignment $m(\cdot)$, assigns a value in $[0, 1]$ to every nonempty subset \mathcal{B} of θ . It is defined by:

$$bel(\mathcal{B}) = \sum_{\mathcal{A} \subseteq \mathcal{B}} m(\mathcal{A}) \quad (9)$$

where the belief function can be viewed as a generalization of a probability function.

8.3 Combination of Evidence

Considering two Basic belief assignments, $m_1(\cdot)$ and $m_2(\cdot)$ and the corresponding belief functions, $bel_1(\cdot)$ and $bel_2(\cdot)$. Let \mathcal{A}_j and \mathcal{B}_k be subsets of θ . Then, $m_1(\cdot)$ and $m_2(\cdot)$ can be combined to obtain the belief mass assigned to $\mathcal{C} \subset \theta$ according to the following formula [48]:

$$m(\mathcal{C}) = m_1 \oplus m_2 = \frac{\sum_{j,k, \mathcal{A}_j \cap \mathcal{B}_k = \mathcal{C}} m_1(\mathcal{A}_j)m_2(\mathcal{B}_k)}{1 - \sum_{j,k, \mathcal{A}_j \cap \mathcal{B}_k = \emptyset} m_1(\mathcal{A}_j)m_2(\mathcal{B}_k)} \quad (10)$$

where the denominator is a normalizing factor, which measures how much $m_1(\cdot)$ and $m_2(\cdot)$ are conflicting.

8.4 Belief Functions Combination

The combination rule can be easily extended to several belief functions by repeating the rule for new belief functions. Thus the sum of n belief functions, $bel_1, bel_2, \dots, bel_n$, can be formed as:

$$((bel_1 \oplus bel_2) \oplus bel_3) \dots bel_n = \bigoplus_{i=1}^n bel_i \quad (11)$$

It is important to note that the basic belief combination formula given above assumes that the belief functions to be combined are independent.

8.5 BBA Based on Single Class Classifiers

In our case, a hypothesis set is defined for each texel in which the image is divided. Within each texel, the hypothesis concerns the possibility that the pixel (i, j) corresponds to an object or not. In other words, we have eight hundred hypotheses for each texel, namely:

$$\theta = \{\theta_1(0, 0), \dots, \theta_1(19, 19), \theta_2(0, 0), \dots, \theta_2(19, 19)\} \quad (12)$$

where $\theta_1(i, j)$ is the belief that the pixel (i, j) of that texel belongs to an object in the environment and $\theta_2(i, j)$ is the belief that the pixel (i, j) doesn't belong to an object.

If we have K benchmarks, we can use K classifiers, each trained using the processes generated by each of the K benchmarks. Each classifier is used as an expert in DS fusion. The goal of the classifiers is to infer from the benchmark where an unknown process comes from.

The classifier C_i provides a probability p_i as output, $i = 1, \dots, K$, where p_i is the probability that the process analyzed by the classifier has been generated by the i -th benchmark. The hypothesis set is given by:

$$\Theta = \{\theta_1, \dots, \theta_K\} \quad (13)$$

where θ_i is the event that the process comes from the benchmark i , $i = 1, \dots, K$. Under this assumption, the expert i provides the probability of the subset $\{\theta_i\}$:

$$m_i(\{\theta_i\}) = p_i \quad (14)$$

If the classifier has been trained with the processes generated by a given benchmark, the probability of the event θ_i shall be distributed to all the other subsets \mathcal{C} of Θ :

$$m_i(\mathcal{C}) = \frac{1 - p_i}{2^K - 1} \quad (15)$$

Then, we consider K single-class classifiers; each classifier is specialized to recognize the workload of a particular benchmark. Under this BBA, the i -th classifier is trained using a set of data produced by the

benchmark i and a set of data not produced by the benchmark i , and it provides as output a real value p_i in the range $[0,1]$, that represents the probability that the current input comes from the benchmark i . Each expert shall assign a belief to the subsets of Θ . Under this BBA, the i -th expert assigns $m_i(\{\theta_i\}) = p_i$ and $m_i(\mathcal{C}) = \frac{1-p_i}{2^K-1}$, $\mathcal{C} \subseteq \Theta$, $\mathcal{C} \neq \{\theta_i\}$.

8.6 Pseudocode

The application of the rule follows the following pseudocode.

```

Foreach classifier
  Foreach classifier
    Translates a set of hypotheses
      to the internally used representation.
    Add a set of hypotheses to the
      evidence and assign a mass to it.
    Combine two evidences.
    
```

9 Case Study: Workload Categorization in the Context of Anomaly Detection Caused by Android Malware

Android mobile devices have become significantly more popular recently, and at the same time, the number of malicious programs operating on them has also grown significantly. As a result, business and academics have given a lot of attention to the security and privacy concerns of Android applications and systems, as Anomaly Detection is crucial due to the increasing dependability of these systems and applications. To this end, behavior-based anomaly detection systems have been developed to identify irregularities brought on by Android malware. These systems analyze these anomalies in data on network traffic, battery temperature, and power usage using machine learning classification algorithms. Hence, we provide in this Section a detailed case study where we show how our proposed approach can be employed within the context of Anomaly Detection caused by Android Malware.

According to a new security report, 4.9 million malware attacks were prevented in the first quarter of 2023 by *Kaspersky mobile security solutions*. Each malware sample must be thoroughly analyzed, which takes time. Malware analysis systems are therefore overloaded by the sheer volume of malware samples. Most newly discovered malware samples are polymorphic versions of already existing malware. By grouping malware samples into different families and then choosing representative samples from each family, we can speed up the analysis of malware. The familial classification of Android malware is difficult, though, for two reasons:

- it is difficult to distinguish between dangerous and legitimate components in the bulk of Android malware samples, which are repackaged versions of popular apps. In fact, 86% of Android malware samples are malicious component-infected repackaged apps [67]. In most cases, just a tiny part of the repackaged apps have dangerous components that have been introduced, which are concealed inside the features of popular programs. System calls [64] and sensitive paths [63] are examples of existing features that make it difficult to distinguish between malware's legitimate and harmful parts;
- the same destructive actions are carried out with multiple implementations by polymorphic Android malware versions that belong to the same family. As a result, malware of this type is readily able to avoid existing classification techniques that look for a precise match to a given specification [11]. As an illustration, two malware samples have various ways of carrying out the same task (i.e., obtaining the device id, phone number, and voice mail number).

Monitoring data is collected and analyzed to determine system health, workload patterns, and metric spaces, which are then utilized to discover anomalies. Furthermore, to test the efficacy of anomaly detection, the detection can be evaluated using different faults to analyze: sensitive API calls, CPU heavy loops, memory leaks, disk I/O errors, and network anomalies:

- sensitive API calls: Android malware typically uses sensitive Application Programming Interface (API) methods, such as *getDeviceId()*, *getLine1Number()*, and *getVoiceMailNumber()*, that operate on sensitive data to carry out harmful actions. For example, in order to gather the consumers' phone numbers, the malware may invoke *getLine1Number()*;
- CPU-intensive loops: Malware causes circular waits and never-ending loops in applications, such as spin lock faults, where CPU resource-intensive operations result in server requests timing out in Android systems and application failures. These injections of fault components result in more computing processes, which need more CPU resources;
- memory leak: this is brought on by assigning heap memory to objects without releasing them, which steadily depletes the system's memory resource and finally causes a system crash. It might take a system with a memory leak a long time before any significant issues arise, making it challenging to identify the issue right away;
- disk I/O error: disk I/O access has a predictable pattern, however, disk access can also be impacted by application-level or hard disk failures in a particular workload pattern. By adding additional disk reading and writing operations to application components, malware trigger this type of these errors;
- network anomaly: Android-based systems and programs are vulnerable to network assaults because rogue scripts may be placed into programs to broadcast schematics that suck up network capacity, saturating servers and leading to service denial. When the injected servlet components are invoked, the malicious code in the application sends UDP packets to any host on the local network.

To this end, the reference architecture shown in Figure 11 for Workload Analysis in Familial Classification of Android Malware, where the architecture is depicted, can be integrated with our proposed Workload Categorization approach using the Dempster-Shafer theory of evidence [48] that we previously described in detail in Section 8. And the hierarchical levels include the following:

- Detection of malicious API calls and FASTA files generation: To identify the APIs, malware applications from different families in reference datasets and benign apps retrieved from Google Play Store are disassembled and analyzed. Furthermore, the frequency of each API call mentioned in both groups is computed independently. API calls are found and documented that are regularly used in malware apps but not in benign apps. Then, based on the suspected Android API types, the API classes are categorized, and these groupings constitute the API class sequence in FASTA format [45]. This structure is required to feed the sequence file into machine learning tools for training and assessment;
- *Multiple Sequence Alignment* (MSA) generation: MSA is created for each app in the family and is used to build a *Profile Hidden Markov Model* (PHMM) using machine learning techniques, which is then trained and utilized as one of the classifiers of unfamiliar Android applications based on a derived score;
- Training and classification: in this layer, the PHMM is trained along with other classifiers such as a Neural Network and k-NN using some reference datasets which consist of malware samples from several families, where these samples are selected from the datasets such that they contain multiple apps from every family. From the selected malware samples, a big percentage of apps is used for training the

classifiers and the rest for testing, and the training is conducted by decompiling the apps and creating the API call list. Then, from the API call list, the suspicious APIs are identified, the final APIs are represented in FASTA format, and the corresponding MSA files are generated. Finally, MSA for all the families are given to the classifiers to create the profile files corresponding to malware families for classification. And the scores generated by each classifier will be combined using the Dempster-Shafer theory of evidence for a final classification;

- Dempster-Shafer fusion: in this layer, the Dempster-Shafer theory of evidence is employed, which combines different measures of evidence on the basis of a finite set of possible hypotheses, which are in our case the scores provided by the classifiers. These scores are also, as a result, the belief values of the assignments represented by the malware families or benign apps that the unknown workload should be classified to. All follow the particular formula presented in Equation 10.

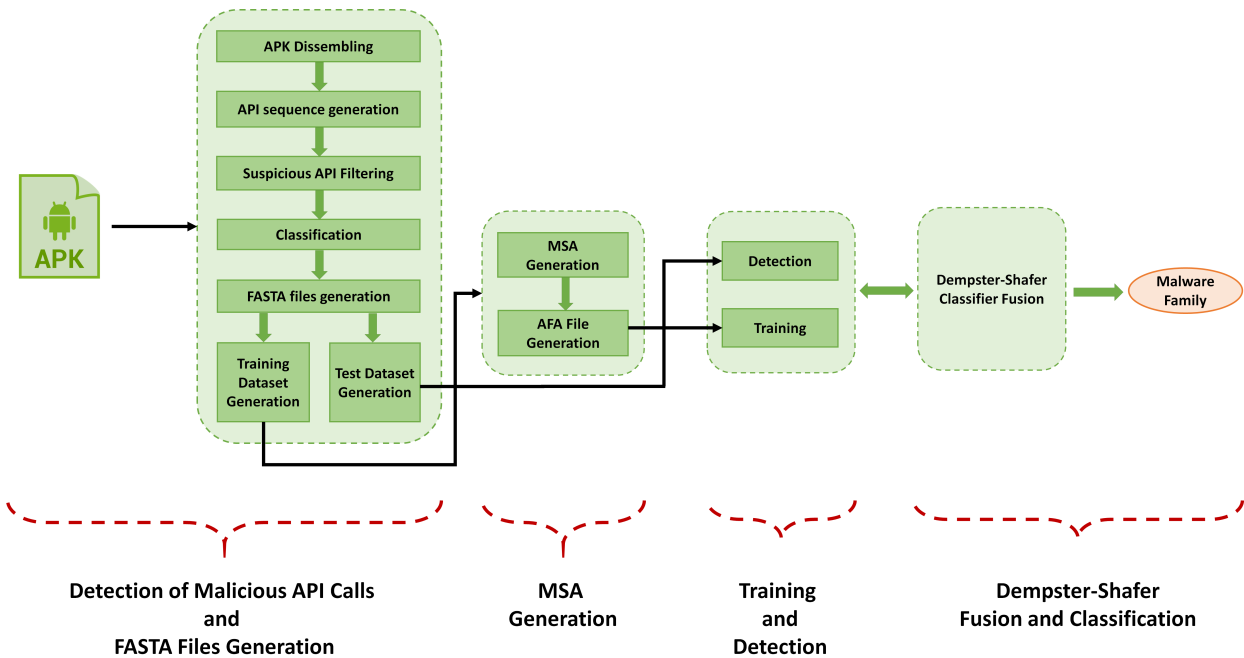


Figure 11: Familial classification of android malware using Dempster-Shafer Fusion

10 Experimental Results and Analysis

First, we specify how the classification experiments that we report in this Section were made. The input data to the classifiers are the metrics acquired during the execution of the benchmarks, namely the memory reference and the resource demand features. The six selected benchmarks are executed three times, each with a different set of input data. We thus obtained the execution sequences that describe the workloads of the different benchmarks. These execution sequences are preprocessed as described in Section 7 and are used to train the classifiers that are so ready to perform the classification. For example, a neural network trained with three different sets of input data but with the same benchmark will become a model of that workload.

The same six benchmarks are then executed with three different sets of input values. Of course, as mentioned before, these executions are completely different from the first ones, but they share the workload for the same benchmark. These executions are classified according to the models derived above.

Let us first consider the result obtained with the memory reference metric. Figure 12 (a) represents the accuracy obtained with the k-NN classifier (e.g., [1]) versus K . From this Figure, we also note that the best accuracy (26.3%) is obtained for $k = 15$. Figure 12 (b) represents the accuracy of the classification with HMM. We used a continuous HMM in which the output distribution is represented by 20 Gaussian mixtures. The used topology is ergodic. This graph shows the classification accuracy as a function of the number of states. The best accuracy is 52.8% and is obtained with six states.

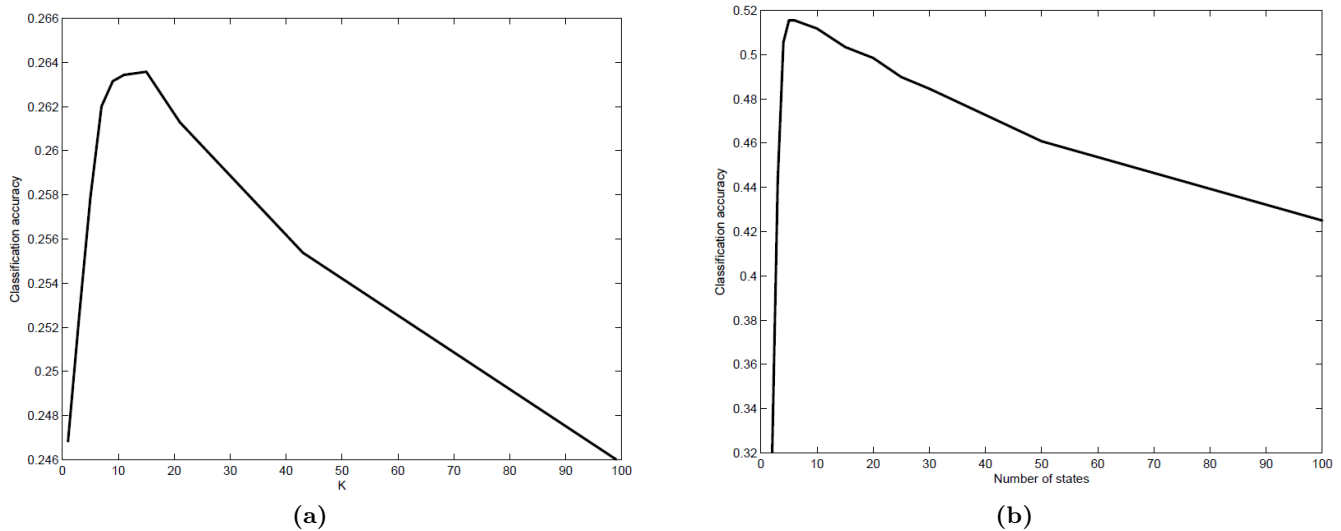


Figure 12: Average classification accuracy of k-NN versus k (a) and HMM versus the number of states (b) using memory reference metrics

Figure 13 shows the accuracy obtained with the neural network. The graph displays the accuracy versus the number of neurons in each hidden layer. Each curve is related to a different number of hidden layers. In particular, curves marked with circles, squares and diamonds are obtained with three, four, and five hidden layers, respectively, and all the other curves are obtained with six up to nine layers. The best accuracy (65.32%) is obtained with four hidden layers and 120 neurons per layer. However, with 50 neurons per hidden layer, the accuracy is around 60% for every number of layers.

Finally, Figure 14 shows the accuracy obtained with the *ARMA model* (e.g., [33]) with $p = 8$ and $q = 4$. The average classification accuracy is 44.15%. The benchmarks are respectively from 1 to 6: *401.bzip2*, *403.gcc*, *458.sjeng*, *471.omnetpp*, *400.perlbench*, and *462.libquantum*.

The Dempster-Shafer data fusion combines the output values from the classifiers. We used different workload metrics and the best classification algorithms according to the results reported in Table 2. The results shown in Figure 15 are obtained by fusing Neural Networks with memory reference features, indicated by NNmr, Neural Networks with resource demand features, indicated by NNrd and HMM with memory references, denoted by HMMmr. In this Figure, the bars show the classification accuracies obtained with, respectively from left to right, the fusion between NNmr and HMMmr, between NNmr and NNrd, between HMMmr and NNrd, and the fusion of HMMmr, NNrd, and NNmr. As shown, the best results are obtained with data fusion between NN with memory reference features and NN with resource demand features, and the obtained accuracy is 79.35%.

The first bar is the data fusion of different classifiers with the same feature (*mr*), while the other bars are related to the usage of two different features (*mr* and *rd*). As shown in Figure 15, the best results are obtained with data fusion between NN with memory reference features and NN with resource demand features, and the

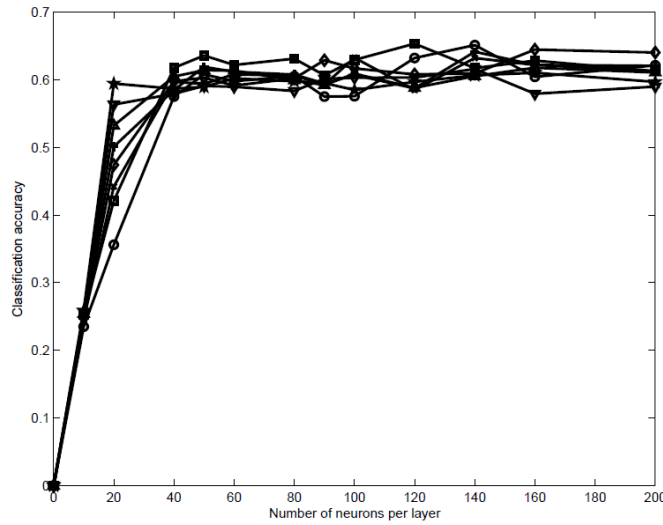


Figure 13: Average classification accuracy of neural networks versus the number of neurons per Layer with memory reference features

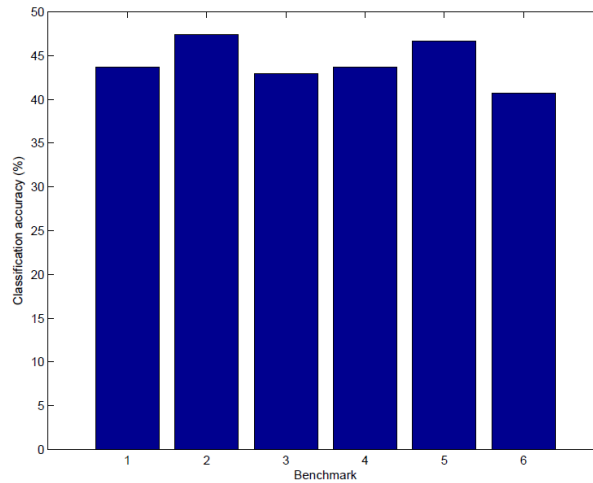


Figure 14: Average classification accuracy obtained with the ARMA model with MA order of 8 and AR order of 4

obtained accuracy is 79.3%. This is an important result that shows that data fusion is effective in boosting classification accuracy with features having different time constants and different computational complexity.

As highlighted so far, using Dempster-Shafer data fusion, we can build a high-quality classifier using lower-quality classifiers. Now, we will show that these high-quality classifiers can be used to find the category of an unknown workload. By testing an unknown execution sequence with the classifier trained on a given workload W , we get an indication of how much the unknown execution sequence can be assigned to the category of the workload W . In fact, if an execution sequence with the workload W is tested with the high-quality classifier trained with a given workload W , the output will be close to one. If an execution sequence with a workload similar to W is tested with the same classifier, the output will be close to 1, and

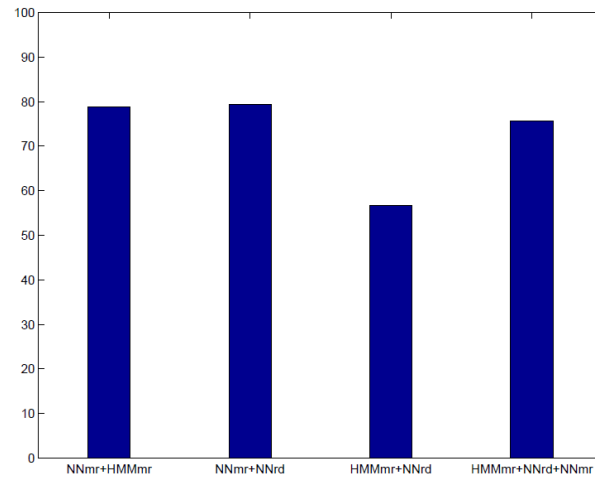


Figure 15: Classification accuracies of the fusion between (from left to right) NNmr and HMMmr, NNmr and NNrd, HMMmr and NNrd, and HMMmr, NNrd and NNmr, respectively

so forth. This property can be used to assign a workload category to an unknown execution sequence. In the experiment described in this Section, we used this property to evaluate distances among the benchmarks from the point of view of the workload they represent. Figure 16 shows a graphical view of the distances among workloads.

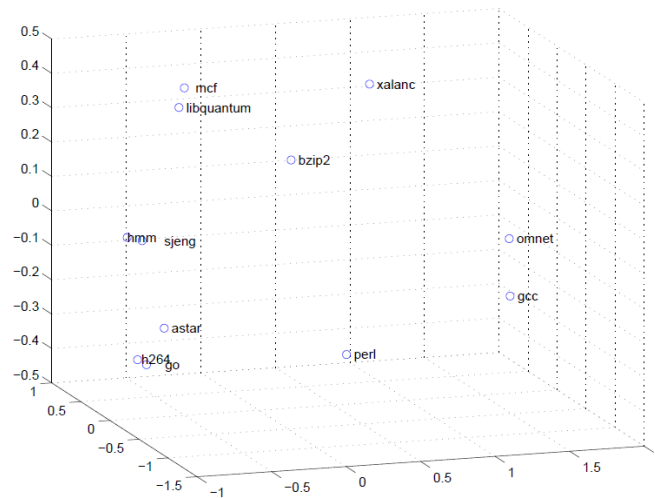


Figure 16: 3D visualization of distances among programs

This Figure shows that *429.mcf* can be given the category of *462.libquantum*, and that *456.hmmer* can be given the category of *458.sjeng* and the workloads of *445.gobmk* and *464.h264ref* are very close.

11 Concluding Remarks and Future Work

In this paper, we deal with the classification of application workloads in a virtualized environment as a means of improving the efficiency and reliability of Cloud-based big data applications. We show that, using lower-quality classifiers, we can build a higher-quality classifier using data fusion algorithms. There may be several applications for this type of classification, from user profiling to malware detection. To this end, we used the SPEC benchmarks that were run in a virtual environment. Different sets of input data were used. The Neural Network classifier gives better results than Hidden Markov Models and K-NN. Final results are obtained by Dempster-Shafer fusion of the Neural Network classification with memory reference and resource demand features, which are workload metrics with completely different time constants. The best classification rate is about 80%.

An obvious extension of the work described in this paper is to use other benchmarks in order to include other workload activities. Also, we plan to further improve the characteristics of our framework by integrating solutions for dealing with novel aspects of massive big data set processing, on top of which workloads may still be defined, such as *data compression techniques* (e.g., [20]), *fragmentation approaches* (e.g., [19]), *privacy-preservation approaches* (e.g., [18]) that, particularly, may be extremely useful when combined with malware detection issues. In addition, we are planning to further enrich our proposed framework by means of emerging big data trends (e.g., [5, 16, 39, 15, 38, 25])

Conflict of Interest: "The authors declare no conflict of interest."

Funding: "This work was partially supported by project SERICS (PE00000014) under the MUR National Recovery and Resilience Plan funded by the European Union - NextGenerationEU."

References

- [1] Altman NS. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*. 1992; 46(3): 175-185. DOI: <https://doi.org/10.1080/00031305.1992.10475879>
- [2] Azmandian F, Moffie M, Dy JG, Aslam JA, Kaeli DR. Workload characterization at the virtualization layer. In: *19th Annual IEEE International Symposium on Modelling, Analysis, and Simulation of Computer and Telecommunication Systems, MASCOTS 2011, 25-27 July 2011, Singapore*. 2011. p.63-72. DOI: <https://doi.org/10.1109/MASCOTS.2011.63>
- [3] Barford P, Crovella M. Generating representative web workload for network and server performance evaluation. In: *Proceedings of the 1998 ACM SIGMETRICS Joint International Conference on Measurement and Modeling of Computer Systems, SIGMETRICS 1998 / PERFORMANCE 1998, 22-26 June 1998, Madison, Wisconsin, USA*. 1998. p.151-160. DOI: <https://doi.org/10.1145/277851.277897>
- [4] Baum LE, Petrie T. Statistical inference for probabilistic functions of finite state Markov chains. *The Annals of Mathematical Statistics*. 1966; 37(6): 1554-1563. DOI: <https://doi.org/10.1214/aoms/1177699147>
- [5] Bellatreche L, Cuzzocrea A, Benkrid S. *F&A*: A Methodology for effectively and efficiently designing parallel relational data warehouses on heterogenous database clusters. In: *Proceedings of the Springer 12th International Conference on Data Warehousing and Knowledge Discovery, DAWAK 2010, 30 June*

- 2010 - 3 September 2010, Bilbao, Spain. 2010. p.89-104. DOI: https://doi.org/10.1007/978-3-642-15105-7_8
- [6] Bhowal P, Sen S, Yoon JH, Geem ZW, Sarkar R. Evaluation of Fuzzy measures using Dempster-Shafer belief structure: A classifier Fusion framework. *IEEE Transactions on Fuzzy Systems.* 2023; 31(5): 1593-1603. DOI: <https://doi.org/10.1109/TFUZZ.2022.3206504>
- [7] Biswas A, Majumdar S, Nandy B, El-Haraki A. Automatic resource provisioning: A machine learning based proactive approach. In: *IEEE 6th International Conference on Cloud Computing Technology and Science, CloudCom 2014, 15-18 December 2014, Singapore.* 2014. p.169-173. DOI: <https://doi.org/10.1109/CloudCom.2014.147>
- [8] Bleikertz S, Vogel C, Groß T. Cloud radar: Near real-time detection of security failures in dynamic virtualized infrastructures. In: *Proceedings of the 30th Annual Computer Security Applications Conference, ACSAC 2014, 8-12 December 2014, New Orleans, USA.* 2014. p.26-35. DOI: <https://doi.org/10.1145/2664243.2664274>
- [9] Bruder G, Steinicke F, Nüchter A. Poster: Immersive point cloud virtual environments. In: *IEEE Symposium on 3D User Interfaces, 3DUI 2014, 29-30 March 2014, Minneapolis, USA.* 2014. p.161-162. DOI: <https://doi.org/10.1109/3DUI.2014.6798870>
- [10] Carlson M. Systems and virtualization management: Standards and the cloud. A report on SVM 2013. *Journal of Network Systems Management.* 2014; 22(4): 709-715. DOI: <https://doi.org/10.1007/s10922-014-9315-7>
- [11] Chen KZ, Johnson NM, D'Silva V, Dai S, MacNamara K, Magrino TR, Wu EX, Rinard MC, Song DX. Contextual policy enforcement in Android applications with permission event graphs. In: *20th Network and Distributed System Security Symposium, NDSS 2013, 24-27 February 2013, San Diego, USA.* 2013.
- [12] Cho Y, Choi J, Choi J. An integrated management system of virtual resources based on virtualization API and data distribution service. In: *ACM Cloud and Autonomic Computing Conference, CAC 2013, 5-9 August 2013, Miami FL, USA.* 2013. p.1-7. DOI: <https://doi.org/10.1145/2494621.2494648>
- [13] Chuang IH, Tsai YT, Horng MF, Kuo YH, Hsu JP. A GA-based approach for resource consolidation of virtual machines in clouds. In: *Springer 6th Asian Conference on Intelligent Information and Database Systems, ACIIDS 2014, 7-9 April 2014, Bangkok, Thailand.* 2014. p.342-351. DOI: https://doi.org/10.1007/978-3-319-05476-6_35
- [14] Cirne W, Berman F. A comprehensive model of the supercomputer workload. In: *Proceedings of the 4th Annual IEEE International Workshop on Workload Characterization, WWC-4 (Cat. No. 01EX538), 2 December 2001, Austin, TX, USA.* 2001. p.140-148. DOI: <https://doi.org/10.1109/WWC.2001.990753>
- [15] Coronato A, Cuzzocrea A. An innovative risk assessment methodology for medical information systems. *IEEE Transactions on Knowledge and Data Engineering.* 2022; 34(7): 3095-3110. DOI: <https://doi.org/10.1109/TKDE.2020.3023553>
- [16] Cuzzocrea A, Martinelli F, Mercaldo F, Vercelli GV. Tor traffic analysis and detection via machine learning techniques. In: *2017 IEEE International Conference on Big Data, IEEE BigData 2017, 11-14 December 2017, Boston, MA, USA.* 2017. p.4474-4480. DOI: <https://doi.org/10.1109/BigData.2017.8258487>
- [17] Cuzzocrea A, Saccà D, Ullman JD. Big data: A research agenda. In: *ACM 17th International Database Engineering & Applications Symposium, IDEAS 2013, 09-11 October 2013, Barcelona, Spain.* 2013. p.198-203. DOI: <https://doi.org/10.1145/2513591.2527071>

-
- [18] Cuzzocrea A, Saccà D. Balancing accuracy and privacy of OLAP aggregations on data cubes. In: *ACM 13th International Workshop on Data Warehousing and OLAP, DOLAP 2010, 30 October 2010, Toronto, Ontario, Canada*. 2010. p.93-98. DOI: <https://doi.org/10.1145/1871940.1871960>
- [19] Cuzzocrea A, Darmont J, Mahboubi H. Fragmenting very large XML data warehouses via K-means clustering algorithm. *International Journal of Business Intelligence and Data Mining*. 2009; 4(3-4): 301-328. DOI: <https://doi.org/10.1504/IJBIDM.2009.029076>
- [20] Cuzzocrea A, Saccà D, Serafino P. A hierarchy-driven compression technique for advanced OLAP visualization of multidimensional data cubes. In: *Proceedings of the Springer 8th International Conference on Data Warehousing and Knowledge Discovery, DaWaK 2006, 4-8 September 2006, Krakow, Poland*. 2006. p.106-119. DOI: https://doi.org/10.1007/11823728_11
- [21] Da Mota B, Tudoran R, Costan A, Varoquaux G, Brasche G, Conrod PJ, Lemaître H, Paus T, Rietschel M, Frouin V, Poline JB, Antoniu G, Thirion B. Generic machine learning pattern for neuroimaging-genetic studies in the cloud. *Frontiers in Neuroinformatics*. 2014; 8: art.31. DOI: <https://doi.org/10.3389/fninf.2014.00031>
- [22] Deng Y, Shen S, Huang Z, Iosup A, Lau RWH. Dynamic resource management in cloud-based distributed virtual environments. In: *Proceedings of the ACM 22nd International Conference on Multimedia, MM 2014, 3-7 November 2014, Orlando, FL, USA*. 2014. p.1209-1212. DOI: <https://doi.org/10.1145/2647868.2655051>
- [23] DiFranzo D, Graves A. A farm in every window: A study into the incentives for participation in the Windowfarm virtual community. In: *ACM 3rd International Web Science Conference, WebSci 2011, 15-17 June 2011, Koblenz, Germany*. 2011. p.1-8. DOI: <https://doi.org/10.1145/2527031.2527042>
- [24] El-Refaey MA, Rizkaa MA. Virtual systems workload characterization: An overview. In: *18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises, WETICE 2009, 29 June 2009 - 1 July 2009, Groningen, The Netherlands*. 2009. p.72-77. DOI: <https://doi.org/10.1109/WETICE.2009.13>
- [25] Genkin M, Dehne F. Autonomic workload change classification and prediction for big data workloads. In: *2019 IEEE International Conference on Big Data, IEEE BigData 2019, 9-12 December 2019, Los Angeles, CA, USA*. 2019. p.2835-2844. DOI: <https://doi.org/10.1109/BigData47090.2019.9006149>
- [26] Gmach D, Rolia J, Cherkasova L, Kemper A. Workload analysis and demand prediction of enterprise data center applications. In: *IEEE 10th International Symposium on Workload Characterization, IISWC 2007, September 27-29, Boston, MA, USA*. 2007. p.171-180. DOI: <https://doi.org/10.1109/IISWC.2007.4362193>
- [27] Goel G, Ganesan R, Sarkar S, Kaup K. Workload analysis for virtual machine placement. In: *IEEE 18th International Conference on Parallel and Distributed Systems, ICPADS 2012, 17-19 December 2012, Singapore*. 2012. p.732-737. DOI: <https://doi.org/10.1109/ICPADS.2012.118>
- [28] Goldberg RP. Survey of virtual machine research. *Computer*. 1974; 7(6): 34-45. DOI: <https://doi.org/10.1109/MC.1974.6323581>
- [29] Gutiérrez-García JO, Ramirez-Nafarrate A. A policy-based agents for virtual machine migration in cloud data centers. In: *IEEE International Conference on Services Computing, 28 June 2013 - 3 July 2013, Santa Clara, CA, USA*. 2013. p.603-610. DOI: <https://doi.org/10.1109/SCC.2013.55>

- [30] Hou HS, Tretter DR, Vogel MJ. Interesting properties of the discrete cosine transform. *Journal of Visual Communication and Image Representation*. 1992; 3: 73-83. DOI: [https://doi.org/10.1016/1047-3203\(92\)90031-N](https://doi.org/10.1016/1047-3203(92)90031-N)
- [31] Hsiao SW, Chen YN, Sun YS, Chen MC. Combining dynamic passive analysis and active fingerprinting for effective bot malware detection in virtualized environments. In: *Proceedings of the Springer 7th International Conference on Network and System Security, NSS 2013, 3-4 June 2013, Madrid, Spain*. 2013. p.699-706. DOI: https://doi.org/10.1007/978-3-642-38631-2_59
- [32] Hu Y, Long X, Wen C. Asymmetric virtual machine scheduling model based on workload classification. In: *IEEE International Conference on Computer Science and Service System, 11-13 August 2012, Nanjing, China*. 2012. p.2231-2234. DOI: <https://doi.org/10.1109/CSSS.2012.554>
- [33] Joachims T. Text categorization with support vector machines: Learning with many relevant features. In: Nedellec C, Rouveirol C. (eds.) *Lecture Notes in Computer Science: Proceedings of the Springer 10th European Conference on Machine Learning, 21-23 April 1998, Chemnitz, Germany*. 1998. p.137-142. DOI: <https://doi.org/10.1007/BFb0026683>
- [34] Kejela G, Esteves RM, Rong C. Predictive analytics of sensor data using distributed machine learning techniques. In: *IEEE 6th International Conference on Cloud Computing Technology and Science, CloudCom 2014, 15-18 December 2014, Singapore*. 2014. p.626-631. DOI: <https://doi.org/10.1109/CloudCom.2014.44>
- [35] Kim BK, Jang JH, Hur KW, Lee JG, Woong Ko Y. Monitoring and feedback tools for realtime workloads for Xen virtual machine. In: *Proceedings of the International Conference on IT Convergence and Security, ICITCS 2011, 14-16 December 2011, Suwon, Korea*. 2011. p.151-161. DOI: https://doi.org/10.1007/978-94-007-2911-7_13
- [36] König JL, Hinze A, Bowen J. Workload categorization for hazardous industries: The semantic modelling of multi-modal physiological data. *Future Generation Computer Systems*. 2023; 141(4): 369-381. DOI: <https://doi.org/10.1016/j.future.2022.11.019>
- [37] Krishna DS, Srinivasi G, Reddy PVGDP. Novel private cloud architecture: A three tier approach to deploy private cloud using virtual machine manager. *Intelligent Decision Technologies*. 2023; 17(2): 275-285. DOI: <https://doi.org/10.3233/IDT-229035>
- [38] Leung CK, Braun P, Hoi CSH, Souza J, Cuzzocrea A. Urban analytics of big transportation data for supporting smart cities. In: *Springer 21st International Conference on Big Data Analytics and Knowledge Discovery, DaWaK 2019, 26-29 August 2019, Linz, Austria*. 2019. p.24-33. DOI: https://doi.org/10.1007/978-3-030-27520-4_3
- [39] Leung CK, Cuzzocrea A, Mai JJ, Deng D, Jiang F. Personalized DeepInf: Enhanced social influence prediction with deep learning and transfer learning. In: *2019 IEEE International Conference on Big Data, BigData 2019, 9-12 December 2019, Los Angeles, CA, USA*. 2019. p.2871-2880. DOI: <https://doi.org/10.1109/BigData47090.2019.9005969>
- [40] Maddodi G, Jansen S, de Jong R. Generating workload for ERP applications through end-user organization categorization using high level business operation data. In: *Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering, ICPE 2018, 09-13 April 2018, Berlin, Germany*. 2018. p.200-210. DOI: <https://doi.org/10.1145/3184407.3184432>

-
- [41] Mahambre S, Kulkarni P, Bellur U, Chafle G, Deshpande D. Workload characterization for capacity planning and performance management in IaaS cloud. In: *2012 IEEE International Conference on Cloud Computing in Emerging Markets, CCEM 2012, October 11-12, Bangalore, India.* 2012. p.1-7. DOI: <https://doi.org/10.1109/CCEM.2012.6354624>
- [42] Manté C. Application of resampling and linear Spline methods to spectral and dispersional analyses of long-memory processes. *Computational Statistics & Data Analysis.* 2007; 51(9): 4308-4323. DOI: <https://doi.org/10.1016/j.csda.2006.05.015>
- [43] Oracle VM VirtualBox. *User Manual.* <https://www.virtualbox.org/manual/> [Accessed 15th January 2023].
- [44] Panneerselvam J, Liu L, Antonopoulos N, Yuan B. Workload analysis for the scope of user demand prediction model evaluations in cloud environments. In: *Proceedings of the 7th IEEE/ACM International Conference on Utility and Cloud Computing, UCC 2014, 8-11 December 2014, London, United Kingdom.* 2014. p.883-889. DOI: <https://doi.org/10.1109/UCC.2014.144>
- [45] Pearson WR, Lipman DJ. Improved tools for biological sequence comparison. *Proceedings of the National Academy of Sciences.* 1988; 85(8): 2444-2448. DOI: <https://doi.org/10.1073/pnas.85.8.2444>
- [46] Ripley BD. Neural networks and related methods for classification. *Journal of the Royal Statistical Society: Series B (Methodological).* 1994; 56(3): 409-437. DOI: <https://doi.org/10.1111/j.2517-6161.1994.tb01990.x>
- [47] Rokach L. Ensemble-based classifiers. *Artificial Intelligence Review.* 2010; 33(1-2): 1-39. DOI: <https://doi.org/10.1007/s10462-009-9124-7>
- [48] Shafer G. *A Mathematical Theory of Evidence.* Princeton University Press; 1976.
- [49] Shao YS, Brooks DM. ISA-independent workload characterization and its implications for specialized architectures. In: *2012 IEEE International Symposium on Performance Analysis of Systems & Software, 21-23 April 2013, Austin, TX, USA.* 2013. p.245-255. DOI: <https://doi.org/10.1109/ISPASS.2013.6557175>
- [50] Sherwood T, Perelman E, Hamerly G, Sair S, Calder B. Discovering and exploiting program phases. *IEEE Micro.* 2003; 23(6): 84-93. DOI: <https://doi.org/10.1109/MM.2003.1261391>
- [51] Singh S, Chana I. Metrics based workload analysis technique for IaaS Cloud. *arXiv [Preprint]* 2014. Doi: <https://arxiv.org/abs/1411.6753>
- [52] SOAP. *Simple object access protocol.* <https://www.w3.org/TR/soap/> [Accessed 15th January 2023].
- [53] Sousa Vieira ME, Suárez-González A, Fernández-Veiga M, López-Ardao JC, López-García C. Model selection for long-memory processes in the spectral domain. *Computer Communications.* 2013; 36(13): 1436-1449. DOI: <https://doi.org/10.1016/j.comcom.2013.06.002>
- [54] SPEC. *The standard performance evaluation corporation.* <http://www.spec.org/> [Accessed 15th January 2023].
- [55] Thakur A, Goraya MS. A workload and machine categorization-based resource allocation framework for load balancing and balanced resource utilization in the cloud. *International Journal of Grid and High Performance Computing.* 2022; 14(1): 1-16. DOI: <https://doi.org/10.4018/IJGHPC.301594>

- [56] VirtualBox Main API. *VirtualBox main API documentation*. <https://www.virtualbox.org/sdkref/> [Accessed 15th January 2023].
- [57] Van Do T. Comparison of allocation schemes for virtual machines in energy-aware server farms. *The Computer Journal*. 2011; 54(11): 1790-1797. DOI: <https://doi.org/10.1093/comjnl/bxr007>
- [58] Vandromme N, Dandres T, Maurice E, Samson R, Khazri S, Moghaddam RF, Nguyen KK, Lemieux Y, Cheriet M. Life cycle assessment of videoconferencing with call management servers relying on virtualization. In: *Proceedings of the 2014 conference ICT for Sustainability, ICT4S-14, 25 August 2014, Stockholm, Sweden*. 2014. p.281-289. DOI: <https://doi.org/10.2991/ict4s-14.2014.34>
- [59] WSDL. *Web services description language*. <https://www.w3.org/TR/wsdl20/> [Accessed 15th January 2023].
- [60] Xiao P, Hu Z, Liu D, Zhang X, Qu X. Energy-efficiency enhanced virtual machine scheduling policy for mixed workloads in cloud environments. *Computers & Electrical Engineering*. 2014; 40(5): 1650-1665. DOI: <https://doi.org/10.1016/j.compeleceng.2014.03.002>
- [61] Xiao L, Chen S, Zhang X. Dynamic cluster resource allocation for jobs with known and unknown memory demand. *IEEE Transactions on Parallel and Distributed Systems*. 2002; 13(3): 223-240. DOI: <https://doi.org/10.1109/71.993204>
- [62] Xu Y, Musgrave Z, Noble B, Bailey M. Workload-aware provisioning in public clouds. *IEEE Internet Computing*. 2014; 18(4): 15-21. DOI: <https://doi.org/10.1109/MIC.2014.38>
- [63] Yang C, Xu Z, Gu G, Yegneswaran V, Porras PA. DroidMiner: automated mining and characterization of fine-grained malicious behaviors in android applications. In: Kutyłowski M, Vaidya J. (eds.) *Lecture Notes in Computer Science: Proceedings of the 19th European Symposium on Research in Computer Security, ESORICS 2014, 7-11 September 2014, Wroclaw, Poland*. 2014. p.163-182. DOI: https://doi.org/10.1007/978-3-319-11203-9_10
- [64] Ying-Dar L, Yuan-Cheng L, Chien-Hung C, Hao-Chuan T. Identifying Android malicious repackaged applications by thread-grained system call sequences. *Computers & Security*. 2013; 39: 340-350. DOI: <https://doi.org/10.1016/j.cose.2013.08.010>
- [65] Zhang J, Figueiredo RJ. Autonomic feature selection for application classification. In: *Proceedings of the IEEE 3rd International Conference on Autonomic Computing, ICAC 2006, 13-16 June 2006, Dublin, Ireland*. 2006. p.43-52. DOI: <https://doi.org/10.1109/ICAC.2006.1662380>
- [66] Zhao X, Yin J, Chen Z, He S. Workload classification model for specializing virtual machine operating system. In: *2013 IEEE 6th International Conference on Cloud Computing, June 28 - July 3, Santa Clara, CA, USA*. 2013. p.343-350. DOI: <https://doi.org/10.1109/CLOUD.2013.144>
- [67] Zhou Y, Jiang X. Dissecting Android malware: Characterization and evolution. *33rd IEEE Symposium on Security and Privacy, SP 2012, 21-23 May 2012, San Francisco, CA, USA*. 2012. p.95-109. DOI: <https://doi.org/10.1109/SP.2012.16>

Alfredo Cuzzocrea

iDEA Lab
University of Calabria
Rende, Italy
&
Department of Computer Science
University of Paris City
Paris, France
E-mail: alfredo.cuzzocrea@unical.it

Enzo Mumolo



Department of Engineering
University of Trieste
Trieste, Italy
E-mail: mumolo@units.it

Islam Belmerabet

iDEA Lab
University of Calabria
Rende, Italy
E-mail: ibelmerabet.idealab.unical@gmail.com

Abderraouf Hafsaoui

iDEA Lab
University of Calabria
Rende, Italy
E-mail: ahafsaoui.idealab.unical@gmail.com

 The Authors.  This is an open access article distributed under the Creative Commons Attribution
4.0 International License (<http://creativecommons.org/licenses/by/4.0/>) 