2015 15th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing

Cloud-based Machine Learning Tools for Enhanced Big Data Applications

Alfredo Cuzzocrea*, Enzo Mumolo† and Pietro Corona†
*ICAR-CNR and University of Calabria, Cosenza, Italy
Email: cuzzocrea@si.dimes.unical.it
†DIA Department, University of Trieste, Trieste, Italy
Email: mumolo@units.it;anpi89@gmail.com

***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 combine 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*. So-called *workload categorization problem* plays a critical role towards improving the efficiency and the reliability of Cloud-based big data applications. Implementation-wise, our method proposes deploying Cloud entities that participate to the distributed classification approach on top of *virtual machines*, which represent classical "commodity" settings for Cloud-based big data applications. Preliminary experimental assessment and analysis clearly confirm the benefits deriving from our classification framework.**

I. Introduction

In this paper, we propose *Cloud-based machine learning tools for enhanced big data applications* (e.g., [29], [6], [17]), where the main idea is that of predicting the "*next*" *workload* occurring against the target Cloud infrastructure via an innovative *ensemble-based* (e.g., [35]) *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., [16]). So-called *workload categorization problem* plays a critical role towards improving the efficiency and the reliability of Cloud-based big data applications (e.g., [47], [48]). Implementation-wise, our method proposes deploying Cloud entities that participate to the distributed classification approach on top of *virtual machines* (e.g., [23]), which represent classical "commodity" settings for Cloud-based big data applications.

Virtualization technology has become fundamental in modern computing environments such as cloud computing [8], [11], [18], [7] and sever farms [41], [19]. 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., [9], [42], [10]). Resource management in the virtualized context can be performed by *classifying the workload of the virtualized application* (e.g., [50]). As a consequence, workload characterization and prediction has been widely studied during past research efforts (e.g., [12], [4]). More recently, some work has been done towards the workload characterization in data center environments [21]. On the other hand, workload modeling and prediction in virtualization environments has been addressed in [20], [22], [3], while wirtualized workload balancing has been addressed in [24], [46].

From the methodological point of view, *workload classification*, 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 [27], 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 in/out intensive. Zhao *et al.* [50] describe a workload classification model based on such a finer classification level. In [49], 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 via using spectral parameters (e.g., [39], [31]). Results on instrumented machines and in simulation shows that *Hidden Markov Model* (HMM) *classifiers* [5] can be used to model memory references created and managed by processes under execution.

In our proposed research, the classification phase is 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* [40]) 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 to each model. Finally, beliefs obtained by means of different classification algorithms are fused using the *Dempster-Shafer rule of evidence combination* [36] in order to derive a higher quality classifier.

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 workload running in virtual-

978-1-4799-8006-2/15 $31.00 © 2015 IEEE
DOI 10.1109/CCGrid.2015.170

ization gives interesting potential applications. For example, if the benchmarks are chosen appropriately, it may be determined what are the main characteristics of processes running in the virtual machine. Another possibility might be to know what are the processes that a given customer typically execute. Other possible applications are in the area of *malware detection* [26]. In this respect, running processes can be monitored to see if their workload is the same or it changes during time. Preliminary experimental assessment and analysis clearly confirm the benefits deriving from our classification framework.

II. 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.

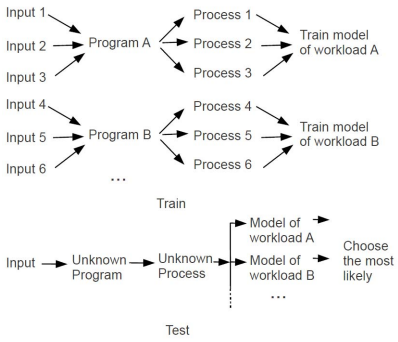

Fig. 1. Training and Testing of Workload Models

We perform workload classification with four classifiers, namely *Neural Networks* (e.g., [34]), *Hidden Markov Models*, *k-Nearest Neighbors* [2] and *ARMA* [28].

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.

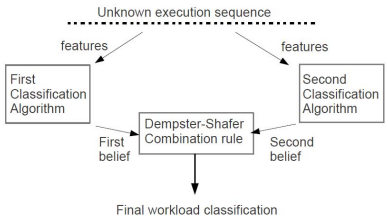

Fig. 2. Data Fusion of Two Classifiers

As it will be shown at the end of the paper, these higher quality classifiers can be use to find the category of an unknown workload.

III. SPEC 2006 Benchmarks

CINT2006 [40] 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. All the benchmarks are provided as source code. The twelve programs included in the benchmark suite, can be grouped in the following classes, according to their functionality: compiler class, game class, compression class, scientific computing class, 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 for evaluating the similarity with 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 is missing. Thus, the reported results have to be considered as preliminary from a general point of view, being valid only whithin computer intensive workloads.

It is important to describe how the input data to 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_ASP.NET-SCRIPT.HTML. 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/INDEX.HTML. Finally, *libquantum* was given the following two additional couples of numbers: (159, 15) and (1413, 17).

In this way, all the benchmarks have at least three inputs which are used for training. Additional input for test has been obtained in similar way.

IV. Virtual Machine Setting

The virtualization infrastructure used in this work is provided by VirtualBox [43]. VirtualBox is an open-source multiplatform virtualization program that provides a rich set of APIs that allow to collect different metrics of the virtualized programs. The complete set of the available APIs is described in [44]. The SDK, which is provided with Virtual Box, allows third parties to develop applications interacting with Virtual Box. Virtual Box is designed in levels. At the bottom we find the VMM (hypervisor). It is the heart of the virtualization engine, monitoring the performance of virtual machines and provides the security and 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). The API level is implemented above these functional blocks.

VirtualBox comes with a web service that, once running, acts as HTTP server, accepts SOAP connections [38] and processes them. The interface is described in a Web Services Description Language (WSDL) file [45]. 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, Perl. In addition to Java and Python, the SDK contains libraries 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. 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. Other interesting features concern the collection of statistical data about the resources usage. The API provides functions for:

- specifying which are the groups of indicators we are interested to (CPU, RAM, Network, Disk);
- setting the measuring range (minimum interval of 1 s);
- setting the frame size for statistics (min, max, average);
- making queries on the usage of a single resource.

V. 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; 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.

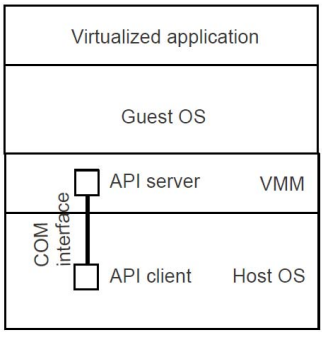

Fig. 3. Acquisition System

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 we report, as an example, a chunk of memory references collected during the virtualized execution of one SPEC benchmark (gobmk).

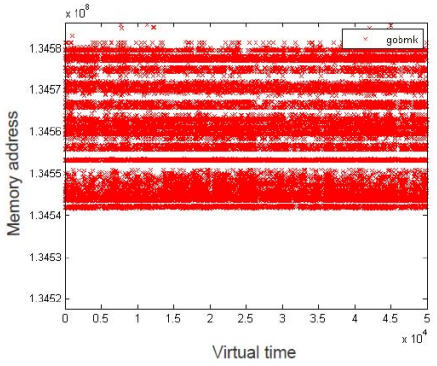

Fig. 4. Memory References Generated by the *gobmk* Benchmark

In Figure 5, we report a chunk of memory reference for another SPEC benchmark (perlbench).

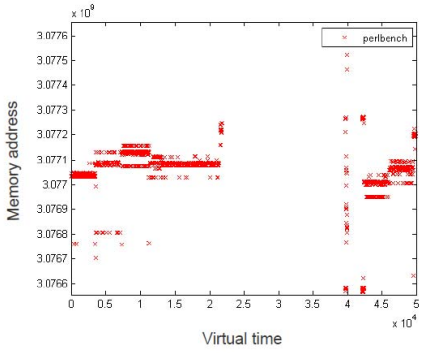

Fig. 5. Memory References Generated by the *perlbench* Benchmark

The data is collected in a *<time stamp><address>* format. Using the IPerformanceCollector API we collect resource demand feature 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;
- memory fragmentation (free memory / total memory).

In Figure 6, 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 7, we report, instead, the amount of free RAM memory available during the execution of the virtualized process.

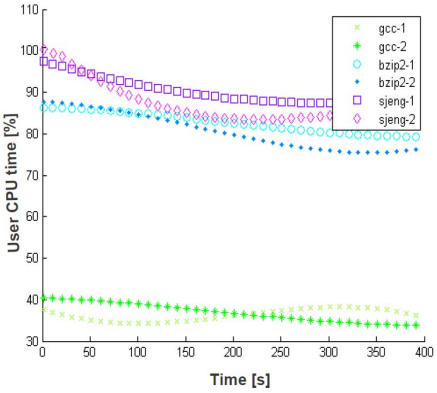

Fig. 6. User CPU Time Feature for Different Programs in Two Different Executions

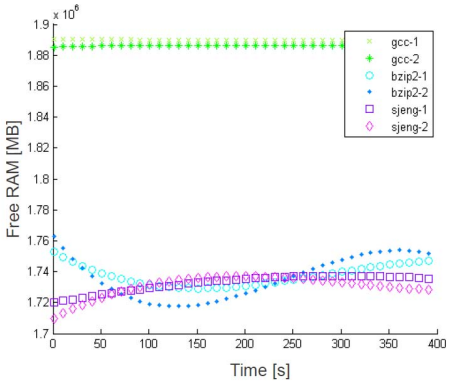

Fig. 7. Free RAM Feature for Different Programs in Two Different Executions

Each curve is related to the execution of a different process in the virtual machine. Also resource demand feature data is acquired in a time-stamp format.

VI. Data Analysis Methodology

It is well known that the initial instructions of a running code is higly non-representative of the steady-state behaviour of the program. In fact the first billions 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 that described in [37] 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 references 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 changes their properties. For these reasons, we used a short time spectral description of the memory references sequence. Thus, the sequences are divided in overlapped analysis frames with a given size, as reported in Figure 8. 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 are divided in overlapped frames and blocks.

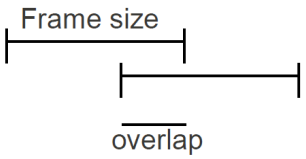

Fig. 8. Analysis Frames

The next step is to perform spectral analysis of the blocks. Among the possible spectral related parameters, we chosen the Discrete Cosine Transform (DCT) representation. DCT is a well-known signal processing operation with important properties [25]. For example, it is useful for reducing the 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 9. A frame of 1024 memory references is plotted in this Figure. On this frame we perform a DCT transform; the first sixteen coefficient are used to obtain a 1024 coefficient vector with zero-padding. By inverse transforming this sequence we obtain the curve plotted in the same Figure 9. 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.

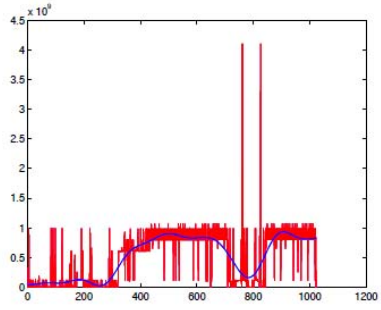

Fig. 9. Signal Reconstruction by Inverse Transforming the First DCT Coefficients

Concerning the resource demand, since there are three types of features in a frame, the DCT is applied separately on 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 components vector, each formed by 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 memory

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

TABLE I. FINAL PARAMETERS SETUP

references and resource demand, is described by an integer number.

Starting from the initial analysis parameters we conducted some experiments of *Neural Network classification*, using three hidden layers and 50 neurons. The final optimal analysis parameters are reported in Table 1.

A. Classification Algorithms

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

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. k-NN instead 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, objects are classified by a majority vote of their neighbors and are assigned to the class most common amongst its k-nearest neighbors. Our k-NN uses the Euclidean distance to represent the closeness to the neighbors of an unclassified object.

2) Neural Network Algorithm: We used a neural networks 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.

3) Hidden Markov Model Algorithm: In 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 which 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 features vectors.

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 autoregressive (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 autoregressive part and q is the order of the moving average part:

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

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.

VII. THE DEMPSTER-SHAFER FUSION

The goal of the Dempster-Shafer theory of evidence [36], is to combine different measures of evidence. At the base of the theory is a finite set of possible hypotheses, say $\theta = \{\theta_1, \ldots, \theta_K\}$.

A. 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:

$$\sum_{\mathcal{A} \subseteq \Theta} m(\mathcal{A}) = 1, \;\; m() = 0$$

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 this means that we can assign belief only to subsets of θ, not to the individual hypothesis.

B. Combination of Evidence

Consider two Basic belief assignments $m_1(.)$ and $m_2(.)$. Then $m_1(.)$ and $m_2(.)$ can be combined to obtain the belief mass assigned to $\mathcal{C} \subset \theta$ according to the following formula [36]:

$$m(\mathcal{C}) = m_1 \bigoplus m_2 = \frac{\sum_{j,k,\mathcal{A}_j \cap \mathcal{B}_k = \mathcal{C}} m_1(\mathcal{A}_j) m_2(\mathcal{B}_k)}{1 - \sum_{j,k,\mathcal{A}_j \cap \mathcal{B}_k = 0} m_1(A_j) m_2(B_k)} \qquad (2)$$

The denominator is a normalizing factor, which measures how much $m_1(.)$ and $m_2(.)$ are conflicting.

C. BBA based on Single Class Classifiers

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 what benchmark an unknown process come from.

The classifier C_i provides a probability p_i as output, $i = 1, \ldots, K$, where p_i is the probability that the process analysed by the classifier has been generated by the ith benchmark. The hypothesis set is given by $\Theta = \{\theta_1, \ldots, \theta_K\}$, where θ_i is the event that the process comes from the benchmark i, $i = 1, \ldots, K$. Under this assumption, the expert i provides the probability of the subset $\{\theta_i\}$:

$$m_i(\{\theta_i\}) = p_i,$$

and, for all the other subset $\mathcal{C}$ of Θ,

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

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.

VIII. Preliminary Experimental Analysis

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 with a given workload W, we get an indication of how much the unknown execution sequence can be assigned 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 one and 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 10 shows a graphical view of the distances among workloads.

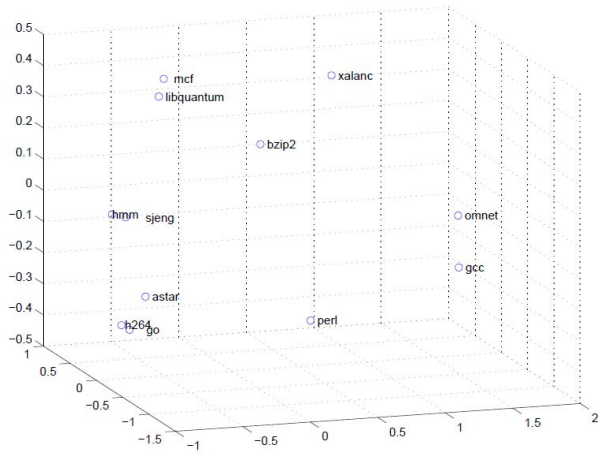

Fig. 10. 3D Visualization of Distances among Programs

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

IX. Concluding Remarks and Future Work

In this paper we deal with the classification of application workloads in a virtualized environment, as to improving the efficency and the 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 of this type of classification, from the user profiling to the detection of malware. 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 be still defined, such as *data compression techniques* (e.g., [15]), *fragmentation approaches* (e.g., [13]), *privacy-preservation approaches* (e.g., [14]) that, particularly, may be extremely useful combined with malware detection issues.

References

[1] P.S.C. Alencar, D.D. Cowan, F. McGarry and R.M. Palmer, *Developing a collaborative cloud-based platform for watershed analysis and management*, IEEE CollaborateCom 2014, pp. 457–459

[2] N.S. Altman, *An introduction to kernel and nearest-neighbor nonparametric regression*, The American Statistician 46(3), 1992, pp. 175-185

[3] F. Azmandian, M. Moffie, J. G. Dy, J. A. Aslam, D. R. Kaeli, *Workload Characterization at the Virtualization Layer*, MASCOTS 2011, pp. 63–72

[4] P.Barford and M.Crovella, *Generating representative web workload for network and server performance evaluation*, ACM SIGMETRICS Performance Evaluation Review 26(1), 1998, pp. 151–169

[5] L.E. Baum, T. Petrie, *Statistical Inference for Probabilistic Functions of Finite State Markov Chains*, The Annals of Mathematical Statistics 37(6), 1966, pp. 1554-1563

[6] A. Biswas, S. Majumdar, B. Nandy and A. El-Haraki, *Automatic Resource Provisioning: A Machine Learning Based Proactive Approach*, IEEE CloudCom 2014, pp. 168–173

[7] S. Bleikertz, C. Vogel, T. Gro, *Cloud radar: near real-time detection of security failures in dynamic virtualized infrastructures*, ACSAC 2014, pp. 26–35

[8] G. Bruder, F. Steinicke, A. Nchter, *Poster: Immersive point cloud virtual environments*, 3DUI 2014, pp. 161–162

[9] M. Carlson, *Systems and Virtualization Management: Standards and the Cloud. A report on SVM 2013*. Journal of Network Systems Management 22(4), 2014, pp. 709–715

[10] Y. Cho, J. Choi, J. Choi, *An integrated management system of virtual resources based on virtualization API and data distribution service*, CAC 2013, p. 26

[11] I.-H. Chuang, Y.-T. Tsai, M.-F. Horng, Y.-H. Kuo, J.-P. Hsu, *A GA-Based Approach for Resource Consolidation of Virtual Machines in Clouds*, ACIIDS 2014, pp. 342–351

[12] W.Cime and F.Berman, *A comprehensive model of the supercomputer workload*, IEEE WWC 2001

[13] A. Cuzzocrea, J. Darmont and H. Mahboubi, *Fragmenting very large XML data warehouses via K-means clustering algorithm*, International Journal of Business Intelligence and Data Mining 4(3/4), 2009, pp. 301–328

[14] A. Cuzzocrea and D. Saccà, *Balancing accuracy and privacy of OLAP aggregations on data cubes*, ACM DOLAP 2010, pp. 93–98

[15] A. Cuzzocrea, D. Saccà and P. Serafino, *A Hierarchy-Driven Compression Technique for Advanced OLAP Visualization of Multidimensional Data Cubes*, DaWaK 2006, pp. 106–119

[16] A. Cuzzocrea, D. Saccà and J.D. Ullman, *Big Data: A Research Agenda*, IDEAS 2013, pp. 198–203

[17] B. Da Mota, R. Tudoran, A. Costan, G. Varoquaux, G. Brasche, P.J. Conrod, H. Lemaître, T. Paus, M. Rietschel, V. Frouin, J.-B. Poline, G. Antoniu and B. Thirion, *Generic Machine Learning Pattern for Neuroimaging-Genetic Studies in the Cloud*, Frontiers in Neuroinformatics 2014(4), 2014

[18] Y. Deng, S. Shen, Z. Huang, A. Iosup, R.W. H. Lau, *Dynamic Resource Management in Cloud-based Distributed Virtual Environments*, ACM Multimedia 2014, pp. 1209–1212

[19] D. DiFranzo, A. Graves, *A farm in every window: a study into the incentives for participation in the windowfarm virtual community*, WebSci 2011, p. 14

[20] M. A. El-Refaey, M. A. Rizkaa, *Virtual Systems Workload Characterization: An Overview*, WETICE 2009, pp. 72–77

[21] D.Gmach, J.Rolia, L.Cherkasova and A.Kemper, *Workload analysis and demand prediction of enterprise data center applications*, IEEE WWC 2007

[22] G. Goel, R. Ganesan, S. Sarkar, K. Kaup, *Workload Analysis for Virtual Machine Placement*, IEEE ICPADS 2012, pp.732–737

[23] R.P. Goldberg, *Survey of Virtual Machine Research*, Computer 7(9), 1974, pp. 34–45

[24] Gutierrez-Garcia, J.O., Ramirez-Nafarrate, *A.Policy-based Agents for Virtual Machine Migration in Cloud Data Centers*, IEEE SCC 2013

[25] H. S. Hou, D. R. Tretter, M. J. Vogel, *Interesting properties of the discrete cosine transform*, Journal of Visual Communication and Image Representation 3(1), 1992, pp. 73–83

[26] S.-W. Hsiao, Y.-N. Chen, Y.S. Sun, M.C. Chen, *Combining Dynamic Passive Analysis and Active Fingerprinting for Effective Bot Malware Detection in Virtualized Environments*, NSS 2013, pp. 699–706

[27] Y. Hu, X. Long, C. Wen, *Asymmetric Virtual Machine Scheduling Model Based on Workload Classification*, CSSS 2012, pp. 2231–2234

[28] T. Joachims, *Text Categorization with Support Vector Machines: Learning with Many Relevant Features*, ECML 1998, pp. 137–142

[29] G. Kejela, R.M. Esteves and C. Rong, *Predictive Analytics of Sensor Data Using Distributed Machine Learning Techniques*, IEEE CloudCom 2014, pp. 626–631

[30] H.-C. Lee, J.-E. Park and M.-J. Lee, *C3ware: A Middleware Supporting Collaborative Services over Cloud Storage*, The Computer Journal 57(2), 2014, pp. 217–224

[31] C. Mant, *Application of resampling and linear spline methods to spectral and dispersional analyses of long-memory processes*, Computational Statistics & Data Analysis 51(9), 2007, pp. 4308–4323

[32] A. Moro, E. Mumolo, M. Nolich, *Ergodic Continuous Hidden Markov Models for Workload Characterization*, ISPA 2009

[33] A. Moro, E. Mumolo, M. Nolich, *Workload modeling using pseudo2D-HMM*, MASCOTS 2009

[34] B.D. Ripley, *Neural Networks and Related Methods for Classification*, Journal of the Royal Statistical Society - Series B 56(3), 1994, pp. 409–456

[35] L. Rokach, *Ensemble-based Classifiers*, Artificial Intelligence Review 33(1-2), 2010, pp. 1–39

[36] G. Shafer, A Mathematical Theory of Evidence, Princeton University Press, 1976

[37] T. Sherwood, E. Perelman, G. Hamerly, S. Sair, B. Calder,*Discovering and Exploiting Program Phases*, IEEE Micro Journal 23(6), 2003, pp. 84–93

[38] SOAP, HTTP://WWW.W3.ORG/TR/SOAP

[39] M.E. Sousa Vieira, A. Surez-Gonzlez, M. Fernndez-Veiga, J.C. Lpez-Ardao, C. Lpez-Garca, *Model selection for long-memory processes in the spectral domain*, Computer Communications 36(13), 2013, pp. 1436–1449

[40] Standard Performance Evaluation Corporation, HTTP://WWW.SPEC.ORG/CPU2006/CINT2006/

[41] T. Van Do, *Comparison of Allocation Schemes for Virtual Machines in Energy-Aware Server Farms*, Computing Journal 54(11), 2011, pp. 1790–1797

[42] N. Vandromme, T. Dandres, E. Maurice, R. Samson, S. Khazri, R.F. Moghaddam, K.K. Nguyen, Y. Lemieux, M. Cheriet, *Life cycle assessment of videoconferencing with call management servers relying on virtualization*, ICT4S 2014

[43] Virtualbox, HTTP://WWW.VIRTUALBOX.ORG/MANUAL

[44] Virtualbox API, HTTP://WWW.VIRTUALBOX.ORG/SDKREF/INDEX.HTML

[45] WSDL, HTTP://WWW.W3.ORG/TR/WSDL20

[46] L. Xiao, S. Chen, X. Zhang, *Dynamic cluster resource allocation for jobs with known and unknown memory demand*, IEEE Transactions on Parallel and Distributed Systems 13(3), 2002

[47] P. Xiao, Z. Hu, D. Liu, X. Zhang and X. Qu, *Energy-efficiency enhanced virtual machine scheduling policy for mixed workloads in cloud environments*, Computers & Electrical Engineering 40(5), 2014, pp. 1650–1665

[48] Y. Xu, Z. Musgrave, B. Noble and M. Bailey, *Workload-Aware Provisioning in Public Clouds*, IEEE Internet Computing 18(4), 2014, pp. 15–21

[49] J. Zhang and R. J. Figueiredo, *Autonomic Feature Selection for Application Classification*, IEEE CAC 2006, pp. 43–52

[50] X. Zhao, J. Yin, Z. Chen, S. He, *Workload Classification Model for Specializing Virtual Machine Operating System*, IEEE CLOUD 2013, pp. 343–350