

An Interference-Aware Application Classifier Based on Machine Learning to Improve Scheduling in Clouds

Vinícius Meyer^a, Dionatrã F. Kirchoff^b, Matheus L. da Silva^c and César A. F. De Rose^d
 Polytechnic School, Pontifical Catholic University of Rio Grande do Sul (PUCRS) - Porto Alegre, Brazil
 vinicius.meyer@edu.pucrs.br

Abstract—To maximize resource utilization and system throughput in cloud platforms, hardware resources are often shared across multiple virtualized services or applications. In such a consolidated scenario, performance of applications running concurrently in the same physical host can be negatively affected due to interference caused by resource contention. This should be taken into account for efficient scheduling of such applications and performance prediction at user level. Nevertheless, resource scheduling in cloud computing is usually based solely on resource capacity, implemented by heuristics such as bin-packing. Our previous work has introduced an interference-aware scheduling model for web-applications considering their resource utilization profile, and to classify applications we applied fixed interference intervals based on common utilization patterns. Although this resulted in placements with better overall results, we observed that some applications with more dynamic workload patterns were wrongly classified with intervals. In this paper, we propose an alternative to the use of intervals and present an interference-aware application classifier for cloud-based applications that deals better with dynamic workloads. Our classifier defines automatically interference levels ranges combining two well-known machine learning techniques: Support Vector Machines and K-Means. Preliminary experiments evaluated the applied machine learning techniques in three quality metrics: Accuracy, F1-Score and Rand Index, observing rates over 80%. The proposed solution creates a workload-aware fine-grained classification that was compared with previous work over different workload scenarios. The results demonstrate that our classification approach improves the placement efficiency by 23% on average.

Index Terms—Interference-Aware Application Classifier, Resource Management, Dynamic Workloads.

I. INTRODUCTION

With the recent core number increment in modern physical machines, cloud-based services can concurrently run many diverse applications even when fully allocated to a single physical machine. In cloud platforms, executing multiple applications on each host is achieved by sharing resources. Unfortunately, multiple cloud-services contending for shared resources generates cross-application interference and can lead to performance degradation [1]. There are several pieces of


evidence showing that interference is related to the performance penalty of application and it may occur depending on the application workload and its variation [2]. In this scenario, cloud-based web services are an example of application type that has dynamic workload. Such applications present an unpredictable intensity variation of resource utilization due to users different usage patterns and periodicity [3].


Traditionally, resource scheduling in cloud computing is based on heuristics such as bin-packing which considers resource capacity [4], [5]. In previous work [6], we have introduced an interference-aware scheduling model for web-applications. This model applies an attraction/repulsion method built upon resource usage profile of each application. However, an interference classification method, which uses fixed thresholds, has been adopted. Such interference ranges were empirically defined. Although this resulted in placements with better overall results, we have observed that applying that method over some applications with high workload variations could lead to an unrepresentative classification estimate. Thus, this approach may disfavor the placement of different types of applications which have dynamic workloads patterns.


In this paper, we propose an interference-aware application classifier which does not require a static threshold setting. Our classification approach defines automatically interference levels ranges from applications combining the benefits of two Machine Learning techniques: Support Vector Machines (SVM) and K-Means. Firstly, it classifies the instances into segments through a training model, dividing interference monitoring metrics effectively into interference classes. Secondly, it discovers their proper interference levels ranges as a consequence of the clustering technique. Finally, it determines application interference levels. This approach combines readily available classification and clustering algorithms as we will demonstrate. The main contributions of the current work are as follows:

- we propose an interference-aware application classifier based on machine learning techniques which sets interference levels ranges automatically, considering dynamic workloads;
- we evaluate the applicability of both machine learning techniques in this problem with three quality metrics obtaining results over 80%;

^a  <https://orcid.org/0000-0001-5893-5878>

^b  <https://orcid.org/0000-0002-6604-8723>

^c  <https://orcid.org/0000-0002-1648-4623>

^d  <https://orcid.org/0000-0003-0070-0157>

- we adapt our group’s placement simulator called CIAPA to work with this new classifier;
- we present experiments comparing our classifier with previous work based on fixed intervals over different workload scenarios;
- we show that our classifier has the potential to significantly improve application placement in consolidated environments.

The rest of this paper is organized as follows. Section II summarizes the concepts used in this study. Section III describes in detail how our proposed solution works and its functionalities. Section IV presents the evaluation methodology adopted to validate our techniques. Section V lists related work. Finally, Section VI depicts our conclusions and future directions.

II. BACKGROUND AND STATE-OF-THE-ART

In this Section, we detail the most important areas to understand the remaining of this study. Firstly, we characterize interference and its impact on performance. Secondly, we introduce the interference-aware scheduling concept. Lastly, we explain the adopted machine learning techniques.

A. Performance Interference

With the advent of resource sharing techniques, physical machines host multiple applications. Even though the use of resource sharing methods, such as virtualization or containerization, provide approaches to fairly share resource between co-hosted applications, when multiple services intensively use a source at the same time, a problem of resource contention will happen. This problem is known as performance interference, and it may lead to severe performance degradation [6].

Virtualization technologies and server consolidation are the main drivers of high resource utilization in modern Data Centers. Combining virtual machines into the same server may lead to severe performance degradation. This performance degradation is known as virtual machine interference. Supporting a higher virtual machine interference may result in a higher consolidation, while strict low interference requirements may demand more resource. Jersark and Ferreto [7] claim that applications are affected by other virtual machines, which use the same resource intensively in the same physical machine. Furthermore, each resource is affected differently. CPU intensive applications led to performance degradation of 14%. Memory and disk I/O intensive applications, the performance degradation were as high as 90%. Therefore, it is clear that performance interference is a problem, and the performance degradation varies depending on the most used resource.

Performance interference affects container-based environments as well. Disk-intensive applications running over containers promote performance degradation that uses different resources intensively. Xavier et al. [8] have tested several combinations of co-hosted workloads. While some of these combinations led to performance degradation of 38%, they could also combine the workloads with no interference.

Cluster systems usually run several applications-often from different users-concurrently, with individual applications competing for access to shared resources such as the file system or the network. Low application performance may be caused by interference from different sources. Shah et al. [9] state that mapping performance data related to shared resources onto time slices can establish the simultaneity of application usage across jobs, which can be indicative of inter-application interference. In some cases, inter-application interference causes performance degradation by up to 50%.

B. Interference-Aware Scheduling

In cloud computing ecosystem, consolidating multiple user applications onto multi-core servers generates interference between co-hosted applications, which impacts application performance. To minimize interference effects and improve applications performance, a common solution is to apply a scheduler which considers interference issues [4], [10]–[13].

Zhu and Tung [10] and Bu et al. [11] present task scheduling strategies that include interference aspects, based on task performance prediction models to realize better workload placement decisions. Proposed models achieve an average error of less than 8% and a speedup of 1.5 to 6.5 times for individual jobs, respectively. Zang et al. [12] and Wang et al. [13] develop interference-aware job scheduling algorithms to estimate the effect of interference among multiple instances of virtualized environments. Results show that proposed scheduling algorithms, on average, reduce the execution time of tasks by 6.5%.

Chen et al. [4] present CloudScope, a system for diagnosing interference for multi-tenant cloud systems. It (re)assigns virtual machines to physical machines and optimizes the hypervisor configuration for different workloads. The interference-aware scheduler improves virtual machine performance by up to 10% compared to the default scheduler.

C. Machine Learning Algorithms

Machine learning techniques are mainly grouped into three categories: (i) reinforcement learning, (ii) supervised and (iii) unsupervised. Reinforcement learning allows a machine to learn its behavior from the feedback received through the interactions with an external environment. Unsupervised machine learning is used to draw conclusions from a given dataset consisting of input data without a labeled target. Supervised machine learning techniques attempt to find out the relationship between input attributes and a target attribute. Supervised methods can further be classified into two main categories: classification and regression. In the regression, the output variable takes continuous values while in classification it takes class labels [14]. In this study two machine learning algorithms have been employed: SVM for classification and K-Means for clustering.

1) *SVM*: Support Vector Machine is a supervised technique. It derives from a hyperplane that maximizes the separating margin between the positive and negative classes in dimensional space. To achieve this, it considers Support Vectors nearest to the minimum cost line. To accommodate curved

lines or polygon regions, it scales the data into higher dimensions for predictions. Aiming to minimize performance degradation in cloud computing, Sotiriadis et al. [15] introduce a virtual machine scheduling algorithm. It applies SVM to classify resource usage. As results, performance degradation has been minimized by 19% and CPU real time has been maximized by 2%. Sant’Ana et al. [16] present a real-time scheduling policy selection algorithm. They evaluated the use of logistic regression and SVM to perform the mapping of running queue job characteristics and machine states. The results show SVM reached a classification accuracy by up to 81%.

2) *K-Means*: Known as a clustering algorithm, K-means is an unsupervised method that attempts to split a given dataset into a fixed number of clusters. Each centroid (k) is an existing data point in the given input dataset. The process of classification and centroid adjustment is repeated until the values of the centroids stabilize. The final centroids will be used to produce the final clustering. Gill et al. [17] propose a resource scheduling technique for holistic management of cloud computing resources. This method uses K-Means for clustering the workloads for execution on different set of resources. Results indicate that authors’ proposed technique is capable of reducing energy consumption by 20.1% while improving reliability and CPU utilization by 17.1% and 15.7% respectively. Xu et al. [18] formulate a generic job scheduling problem for parallel processing of big data in heterogeneous clusters and design a K-Means based task scheduling algorithm, referred to as KMTS. Simulation results show that KMTS improves execution performance by 25% and 30% on average in single job scheduling and parallel job scheduling, respectively, over existing methods.

III. INTERFERENCE-AWARE APPLICATION CLASSIFIER

Most of the state-of-the-art related studies employ prediction models to address interference scheduling issues in cloud environments. In previous work, we have presented placement policies that deal with those aspects. Considering workloads behavior, we created a static classification to evaluate our strategies. However, neither our previous work nor earlier mentioned studies quantify interference levels from applications without establishing static ranges. To tackle this topic, we propose an interference-aware application classifier based on machine learning techniques. Our classifier receives monitored metrics from applications and automatically outcomes their interference levels, without setting thresholds. Figure 1 depicts a detailed overview of how the classifier works.

To perform the proposed classifier, two machine learning algorithms have been employed: one for classification (SVM) and one for clustering (K-Means). First, the interference metrics collected from the target application are consumed by SVM and those metrics are separated into resource classes: Memory, CPU, Disk, Network and Cache. Subsequently, K-Means quantifies all resource classes and outcomes their interference levels. Both machine learning algorithms use a training dataset, previously defined, to support their decisions.

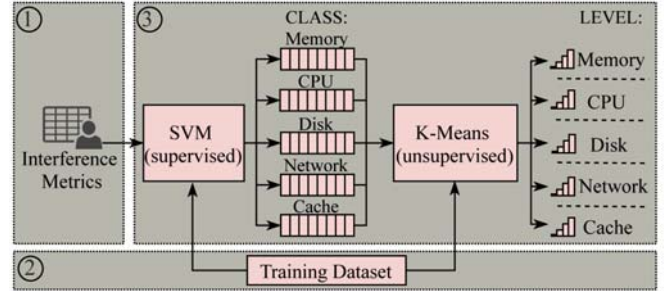


Fig. 1. Overall Classifier Architecture: Block 1 represents interference metrics collected from the application as input data; Block 2 depicts the Training Dataset supporting machine learning techniques decisions; Block 3 illustrates the entire classification process, combining SVM and K-Means as well as their outcomes.

In this Section, we explain the overall functionality of how the proposed classifier works, including its dependencies and capabilities. Firstly, we introduce how the collecting of interference metrics has been performed. Secondly, we explain how the training dataset has been built. Lastly, we describe the classifier process.

A. Collecting of Interference Metrics

To characterize the interference generated by each application, we used a tool called IntP [8]. This tool profiles the application during runtime, returning interference the application generates on each resource subsystem. Moreover, IntP returns the interference metrics, every second, where the higher the metric is, the more interference the application being profiled generates. IntP is a tool developed within the operating system level. It is supported to be executed inside all low-level components without any intrusion that could alter the metrics collected. This tool is partitioned in modules that are responsible for each type of access on a specific resource inside the infrastructure level. IntP outcomes the percentage on how much the monitored application uses the hardware resources like CPU, disk, memory, network, and cache. More specifically, it returns the percentage interference of following metrics:

- *netp* - physical network;
- *nets* - network queue;
- *blk* - disk;
- *mbw* - memory bandwidth;
- *llcmr* - last-level cache miss rate;
- *llcocc* - last-level cache occupation;
- *cpu* - CPU utilization;

B. Training Dataset

To taking advantage of selected machine learning techniques, as mentioned before, it is mandatory to have a variety of data as input to train the models. However, no available datasets were found in the literature with cross-application interference traces. To tackle this issue, a tool called Node-Tiers¹, has been used. It is a multi-tier benchmark that allows

¹<https://github.com/uillianluz/node-tiers>

fine-grained personalization of resource utilization. It stresses the computer system in various selectable ways. It was designed to exercise various physical subsystems of a computer through web requests. It can be useful to observe performance changes across different operating system releases or types of hardware.

In order to keep a data history from each class of interference, we need to stress the main resource classes and store interference metrics from it. To better explain, let us take an example: To collect CPU interference metrics, Node-Tiers has been set with *cpu* parameter, which means only the CPU will be stressed. To collect Cache class, the *cache* parameter has been set, and so on. We have created five major classes of interference: *Memory*, *CPU*, *Disk*, *Network* and *Cache*. Thus, 10,000 samples have been collected from each class of interference, resulting in a dataset with 50,000 samples.

The interference collecting phase has been performed over a Dell PowerEdge R740xd equipped with: 2x Intel Xeon Gold 5118 Processor, 300GB of DDR4 RAM Memory, 1TB Hard Drive and 4x Gigabit Ethernet Interface. The adopted operating system is Ubuntu Server 16.04 LTS (Xenial Xerus).

C. Classifier Process

The main idea of the proposed classifier is to discover interference levels from a given application based on its workload behavior, within a given period, without user intervention.

The target application is monitored with the IntP tool, every second. After a while, the classifier gets interference metrics collected from IntP and uses as input data. IntP returns the percentage of seven resources (seen in Subsection III-A) that receive interference from an application every second. Subsequently, the SVM technique trains the model with the dataset previously mentioned and returns the classification results. Since SVM is a supervised technique, it uses labeled data from a training dataset to label the new data. After classifying that application into target classes (Memory, CPU, Disk, Network, and Cache), those classes are stored in queues and become K-Means input data. We have set four possible levels: Absent, Low, Moderate, and High. When there is no interference activity over some of the classes, the classifier understands it as Absent. When there is interference activity, it is sent as input data to K-Means.

K-Means, previously trained, defines the interference levels of each resource class. K represents classes division: Low, Moderate, and High. Hence, its value has been set to 3 ($k=3$).

When a cycle of monitoring ends, interference metrics are fed into the classifier as input and, as an outcome, it returns the level of application's interference over each hosted resource. Those interference metrics refers to the interval of time that was previously monitored, considering the target application has dynamic and not known workload (i.e. web services).

Since interference is classified from a given time slice, every monitoring cycle is repeated, and the classification occurs based on the training dataset. SVM technique trains its model in the first monitoring cycle. After that, the model does not change, once the training dataset is always the same. K-Means

algorithm finds its centroids based on training dataset at the beginning of its execution. Once its centroids are found, they are always the same since training dataset is the same, as well. The proposed classifier has been implemented adopting the free statistical software tool R². Table I lists the R packages adopted to execute each of the algorithms discussed in this paper.

TABLE I
MACHINE LEARNING TECHNIQUES AND THEIR R PACKAGES.

Technique	Package	Citation
SVM	e1071	Meyer et al. [19]
K-Means	stats	R Core Team [20]

Even though R has been chosen in this study, the model design is not limited to this specific tool, other software tools, such as Keras³ for Python or Weka⁴ for Java, could also potentially be used. One factor in choosing (or dismissing) a machine learning platform is its coverage of existing algorithms [21]. R provides flexibility for implementing several types of model architectures.

Considering there are different machine learning techniques in the literature, depending on the chosen one, there are different parameters to be set. In order to: (i) not over- or under-fitting the training model; (ii) to eliminate the user responsibility of setting these parameters; and (iii) to find the best set of parameters for machine learning techniques, *caret*⁵ package has been used. This package provides a standard syntax to execute a variety of machine learning methods, thus simplifying the process of systematically comparing different algorithms and approaches.

Since our classifier uses application behavior through interference metrics, this model can be executed with a diversity of interference input data. In this case, we have elected IntP, but it is not the only tool that could be used. If for some reason, we decide to use a different tool as interference-profiler (i.e. PAPI⁶), some modifications have to be carried out. It is important mentioning that a machine learning technique may outcome better results than another, depending on the data quality. All files, including source codes and results, are available in a GitHub⁷ repository.

IV. EVALUATION

In this Section, an analysis is performed over the training dataset, as well as the model validation with quality metrics and a comparison with previous work.

²<https://www.r-project.org/>

³<https://keras.io/>

⁴<https://www.cs.waikato.ac.nz/ml/weka/index.html>

⁵<https://cran.r-project.org/web/packages/caret/index.html>

⁶<https://icl.utk.edu/papi/>

⁷<https://github.com/ViniciusMeyer/pdp2020>

A. Dataset Analysis

For data analysis, the machine learning technique processes the fed dataset to the system and generates a set of descriptive statistics on the included features (interference metrics). Besides several metrics, these techniques support statistics on configurable slices of the data and cross-feature statistics such as Correlation between features. Here, Correlation refers to the statistical measure of the relationship between interference metrics.

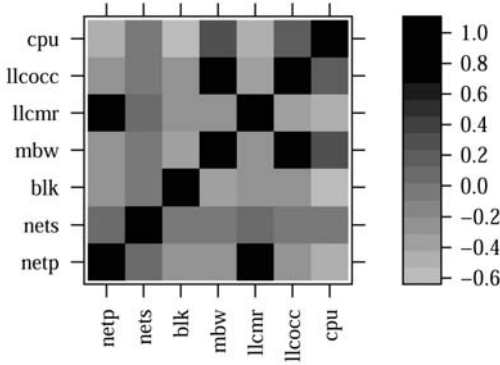


Fig. 2. Correlation of Dataset Interference Classes.

Figure 2 presents Correlation between interference metrics collected with IntP, above mentioned. By looking at these feature statistics, users can gain insights into the shape of the dataset, such as:

- Different assets moving in the same direction are positively correlated; if they move together exactly, they are perfectly positively correlated.
- Negatively correlated returns move in opposite directions. Series that move in exactly opposite directions are perfectly negatively correlated.
- Uncorrelated returns have no relationship to each other and have a correlation coefficient of close to zero.

In our dataset, there are strong positive correlations between some metrics, such as *llcocc* and *mbw*. On the other hand, there are negative correlations between metrics, such as *cpu* and *blk*. It means that: (i) while Cache is used, memory bandwidth is used as well; and (ii) while CPU is consumed, the disk is almost not used. This information is important for K-Means (clustering phase) since it uses those data to train and find the optimal centroids arrangement (interference interval levels). It is worth noting that information comes from the training dataset, and if we change it, the correlation between resources probably will have different behavior, strongly depending on the data.

B. Model Validation

A model, which benefits from machine learning techniques, has a validation step that is used to ensure the proposed models have a good quality of classification or clustering. We validate that a model is safe to serve when a simple premise is reached: the quality metrics have to achieve reasonable rates.

For this purpose, we have determined two classification quality measures [22]: (i) Accuracy is the most common and simplest measure to evaluate a classifier. It is defined as the degree of right predictions of a model (or conversely, the percentage of miss-classification errors) and (ii) F1-Score (or F-Measure), that makes a relation between Precision and Recall metrics. We evaluate the SVM algorithm repeating a 5-fold stratified cross-validation 10 times, with different randomly-selected partitions. This process chooses the model with the best validation score.

For clustering, we have defined Rand Index [23] (or Rand Measure) as a quality measure. Rand Index is a measure of the similarity between two data clustering. It has become the index of choice in comparing the agreement between two separate partitions of the same dataset. This measure adjusts for chance agreement and is not restricted to comparing partitions with the same number of segments. Complete independence between the two partitions yields a Rand Index of essentially zero. Complete association yields an index of 1.0. From a mathematical standpoint, this index is related to the accuracy but is applicable even when class labels are not used.

All quality measures range between 0 and 1. The higher the measured value, the better their quality. These metrics are shown in Table II.

TABLE II
QUALITY MEASURES OF MACHINE LEARNING TECHNIQUES.
(- NOT APPLICABLE)

Measure	SVM	K-Means
Accuracy	0.97	-
F1-Score	0.98	-
Rand Index	-	0.82

Within our study case, all quality metrics have a good rate. This means that both machine learning techniques of the proposed classifier induces a good training quality.

C. Comparison Against Previous Work

To evaluate the proposed classifier, we have compared it against previous work [6], using a tool called CIAPA⁸. This tool uses an interference cost function to analyze the placement of the applications. This function gives the total interference cost of running an application, represented by their interference set I' . The interference level for each resource is denoted as follows:

$$g(I'_{res}) = \{I \mid I \in I'_{res}, I > 1\} \quad (1)$$

Where $res = \{CPU, memory, disk, cache, network\}$. The function g denoted in Equation 1 returns a set of values that are greater than 1. All resource interference metrics are measured and allocated into an interval. Depending on the interval which they are set, the cost value varies according Table III.

⁸<https://uillianluz.github.io/ciapa>

TABLE III
PERFORMANCE DEGRADATION GENERATED BY RESOURCE INTERFERENCE
INTRODUCED BY LUDWIG ET AL. [6]

Level	CPU	Memory	Disk	Network	Cache
Absent	1.00	1.00	1.00	1.00	1.00
Low	1.03	1.07	1.12	1.05	1.07
Moderate	1.15	1.62	1.82	1.32	1.18
High	1.33	1.74	2.25	1.57	1.26

The total interference cost is finally given by the multiplication of the cost of each resource, which is calculated by using the function seen in Equation 2.

$$f_i(I') = f_i'(I'_{cpu}) * f_i'(I'_{mem}) * f_i'(I'_{disk}) * f_i'(I'_{cache}) * f_i'(I'_{net}) \quad (2)$$

CIAPA tries to minimize the total cost by testing all possible combinations of applications per host. Moreover, the total cost of a placement is given by the average costs of each host.

To execute dynamic workloads, we have elected three different applications which can perform workload variations. The first one is a QoS-oriented e-commerce benchmark called Bench4Q [24]. This application has features to deduce a controllable and flexible representation of complex session-based workloads and to simulate authentic customer behavior. The second application is a database benchmark developed to evaluate database performance for workloads similar to those of Facebook's production, named LinkBench⁹. LinkBench is highly configurable and extensible. It can be reconfigured to simulate a variety of workloads and plugins can be written for benchmarking additional database systems. Last application is a decision support benchmark called TPC-H¹⁰. It evaluates the performance of various decision support systems by the execution of sets of queries against a standard database under controlled conditions.

Different workload variations can change the application's behavior in terms of resource usage and performance. In order to variate applications behavior, four workload patterns have been set for each application. We have set up the following workloads: *Increasing*, *Periodic*, *Decreasing* and *Constant*. This idea has been inspired by Iqbal et al. [3] study, where *Increasing* starts with a low load and gradually goes to a high load. *Periodic* has continuously high-to-low and low-to-high variations loads. *Decreasing* is the opposite of *Increasing*, it starts with a high load and gradually goes to a low load. *Constant* keeps always the same workload.

Figure 3 presents the execution of Bench4Q with increasing workload jointly with its corresponding classification of interference levels. Note that while CPU (A) and cache (C) are more stressed, memory (B), disk (D) and network (E) are less stressed. Disk and network resources undergo low interference rates, on average less than 2% and 5%, respectively. In this case, network classification is lead to absent-level while disk is

⁹<http://github.com/facebookarchive/linkbench>

¹⁰<http://www.tpc.org/tpch/>

categorized as low-level. However, although memory generates low interference rates as well (less than 25% on average), its usage is classified as high-level. It means that not only each isolated resource is considered in the classification, but their entire combination.

To evaluate our proposed classifier, two tests scenarios were created:

- **Scenario 1:** Each of the three applications was submitted to 4 workload patterns, resulting in 12 different variations. These variations have been tested over different numbers of hosts, as follows: 4, 6, 8, 10 and 12.
- **Scenario 2:** In this scenario, the variations from scenario 1 were duplicated, resulting in 24 variations. These workloads have been tested over different numbers of hosts, as follows: 8, 10, 12, 16 and 24.

We have applied two classification methods on each workload: (i) Our classifier and (ii) Ludwig et al. classification. Classification outcomes from each scenario were inserted into CIAPA and the results are presented in Figure 4. It is interesting to note that in all experiments our solution reaches lower placement costs, creating a workload-aware fine-grained classification. In both scenarios, with the smallest number of hosts, the largest cost difference has occurred. These were the cases that happened more cross-application interference. Resulting in greater performance degradation among co-located applications. As long as the number of hosts grows, the difference between placement costs decreases. Therefore, resource concurrency among co-hosted applications tends to decrease as well. Only with the largest number of hosts, both classification methods, in both scenarios, have achieved the same values. This happened because each host ran only one application instance. Since there is no incidence of cross-application interference, the cost function has been led to the minimum.

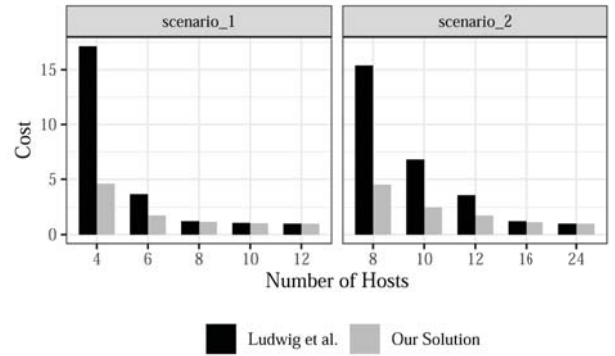


Fig. 4. Comparison of Average Placement Costs.

V. RELATED WORK

Consolidating multiple applications in physical machines, with virtualization techniques, has become a standard to cloud providers. This consolidation, however, may result in performance-related problems such as resource interference.

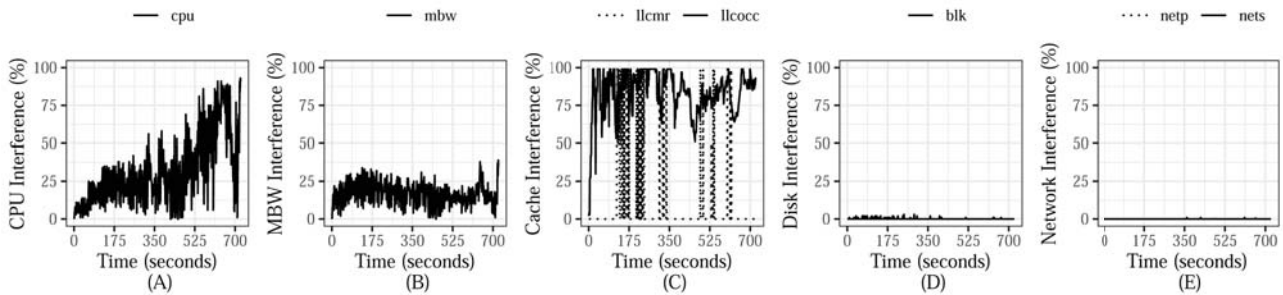


Fig. 3. Bench4Q execution with increasing workload. Interference levels classification: CPU-moderate, Memory-high, Cache-high, Disk-low, Network-absent.

In order to reduce the effects of such problems, Ludwig et al. [6] propose placement algorithms based on interference and affinity policies and evaluate them for different workload scenarios. As results, they achieve a reduction in response time of 10% when compared to interference strategies and up to 18% when considering only affinity strategies.

Jersak et al. [7] devised a simple interference model to be used as a proof of concept with its proposed virtual machine placement strategy. In this model, the level of interference is defined as a function of the number of virtual machines co-executing in a physical machine. So, the model considers that the higher the number of virtual machines, the higher the interference level will be. They also have verified that each resource is affected differently. An example is the addition of many CPU intensive applications generates performance degradation of 14%. On the other side, for memory and disk I/O intensive applications, the performance degradation was as high as 90%.

Hood et. al. [25] quantified the costs of contention for resources in the memory hierarchy by utilizing a differential performance analysis methodology. Intending to measure the impact of shared multicore resources and to consider how contention affects running applications it was compared MPI processes running in different patterns, thus isolating the effects of resource sharing. Considering the Non-Uniform Memory Access (NUMA) and Uniform Memory Access (UMA) processors core configurations, the performance difference between two configurations is translated into a "penalty" for increased sharing of a resource.

Performance degradation and predictability may be caused by running multiple applications with multiple cores sharing the last level on-chip cache (LLC). Aiming to mitigate these problems C.-J. Wu and Martonosi [26] quantified the significant degree of intra-application interference in current workloads and systems. The authors propose insertion policies strategies for reducing each major sources of LLC interference which improves user IPCs in 19%. Furthermore, the authors propose a prefetch manager which eliminates as many as 25% of LLC misses compared to the system default.

Dorier et. al. [27] proposed the Cross-Application Layer for Coordinated I/O Management (CALCioM) framework. This framework integrates the interfering, serializing, and

interrupting coordination strategies aiming to mitigate the I/O interferences in HPC systems by exploiting cross-application coordination. Through CALCioM, the most appropriate one for a targeted machine-wide efficiency, which means that an application will pause its I/O activity for the benefit of another application.

Jin et al. [28] classified applications in three SLLC access classes called (i) cache-pollution, (ii) cache-sensitive and (iii) cache-friendly. These classes were further used to propose a virtual machine placement strategy to alleviate interference by co-locating applications with compatible SLLC access profiles. Their work claimed that cache-pollution applications should be preferably co-located with cache-friendly applications rather than being co-located with cache-sensitive ones.

Alves et al. [29] propose a multivariate and quantitative model able to predict cross-application interference level that considers the number of concurrent accesses to SLLC, DRAM and virtual network, and the similarity between the amount of those accesses. An experimental analysis of proposed prediction model by using a real reservoir petroleum simulator and applications from a well-known HPC benchmark showed that this model could estimate the interference, reaching an average and maximum prediction errors around 4% and 12%, and achieving errors less than 10% in approximately 96% of all tested cases. The authors claim that the cross-application interference problem is related to the amount of simultaneous access to several shared resources, revealing its multivariate and quantitative nature.

VI. CONCLUSION AND FUTURE DIRECTIONS

Previous and related work show that interference generated by co-hosted applications concurrently accessing shared resources in virtualized cloud environments can lead to significant performance degradation. Detecting the interference source is the first step to reduce resource contention. In this study, we have presented an interference-aware application classifier, focusing on dynamic workloads. Our solution combines two well-known machine learning techniques, namely Support Vector Machines (SVM) and K-Means, to automatically identify stressed resources by an application without the need of predefined intervals, that have to be empirically

tuned, and are unable to properly handle applications which have dynamic workloads patterns.

In preliminary experiments, we evaluated the proposed machine learning techniques in three quality metrics: Accuracy, F1-Score and Rand Index, observing rates over 80%. Therefore, these techniques are able to effectively indicate the level of interference the application generates for each resource so that the proposed classifier creates a workload-aware fine-grained classification. In addition, we have compared our solution with a different classification approach, from previous work, and have verified that our classification method results in more efficient placement decisions. For the tested scenarios, placement efficiency improved by 23% on average, reducing resource consumption and also performance degradation at application level.

In future work, we expect to evaluate the influence of applications interference over time. We are interested in investigating the use of time-series segmentation algorithms, such as sliding windows, to find the best intervals to dynamically make placement decisions.

ACKNOWLEDGEMENT

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Finance Code 001, Forense Science National Institute of Science and Technology (INCT) financed by CNPq Brazil. This work has been partially supported by the project "GREEN-CLOUD: Computação em Cloud com Computação Sustentável" (#16/2551-0000 488-9), from FAPERGS and CNPq Brazil, program PRONEX 12/2014.

REFERENCES

- [1] Y. Amannejad, D. Krishnamurthy, and B. Far, "Detecting performance interference in cloud-based web services," in *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, May 2015, pp. 423–431.
- [2] M. G. Xavier, M. V. Neves, F. D. Rossi, T. C. Ferreto, T. Lange, and C. A. F. De Rose, "Performance evaluation of container-based virtualization for high performance computing environments," in *21st Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, Feb 2013, pp. 233–240.
- [3] W. Iqbal, A. Erradi, and A. Mahmood, "Dynamic workload patterns prediction for proactive auto-scaling of web applications," *Journal of Network and Computer Applications*, vol. 124, pp. 94 – 107, 2018.
- [4] X. Chen, L. Rupprecht, R. Osman, P. Pietzuch, F. Franciosi, and W. Knottenbelt, "Cloudscope: Diagnosing and managing performance interference in multi-tenant clouds," in *2015 IEEE 23rd International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, Oct 2015, pp. 164–173.
- [5] V. Meyer., M. G. Xavier., D. F. Kirchoff., R. da R. Righi., and C. A. F. D. Rose., "Performance and cost analysis between elasticity strategies over pipeline-structured applications," in *International Conference on Cloud Computing and Services Science (CLOSER)*, 2019, pp. 404–411.
- [6] U. L. Ludwig, M. G. Xavier, D. F. Kirchoff, I. B. Cezar, and C. A. F. De Rose, "Optimizing multi-tier application performance with interference and affinity-aware placement algorithms," *Concurrency and Computation: Practice and Experience*, p. e5098, 2019, e5098 cpe.5098.
- [7] L. C. Jersak and T. Ferreto, "Performance-aware server consolidation with adjustable interference levels," in *31st Annual ACM Symposium on Applied Computing*, ser. SAC '16. ACM, 2016, pp. 420–425.
- [8] M. G. Xavier, "Data processing with cross-application interference control via system-level instrumentation," Ph.D. dissertation, Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil, 2019.
- [9] A. Shah, F. Wolf, S. Zhumatiy, and V. Voevodin, "Capturing inter-application interference on clusters," in *IEEE International Conference on Cluster Computing (CLUSTER)*, 2013, pp. 1–5.
- [10] Q. Zhu and T. Tung, "A performance interference model for managing consolidated workloads in qos-aware clouds," in *2012 IEEE Fifth International Conference on Cloud Computing*, June 2012, pp. 170–179.
- [11] X. Bu, J. Rao, and C.-z. Xu, "Interference and locality-aware task scheduling for mapreduce applications in virtual clusters," in *Proceedings of the 22Nd International Symposium on High-performance Parallel and Distributed Computing*, ser. HPDC '13. New York, NY, USA: ACM, 2013, pp. 227–238.
- [12] W. Zhang, S. Rajasekaran, T. Wood, and M. Zhu, "Mimp: Deadline and interference aware scheduling of hadoop virtual machines," in *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, May 2014, pp. 394–403.
- [13] K. Wang, M. M. H. Khan, N. Nguyen, and S. Gokhale, "Design and implementation of an analytical framework for interference aware job scheduling on apache spark platform," *Cluster Computing*, vol. 22, no. 1, pp. 2223–2237, Jan 2019.
- [14] S. Athmaja, M. Hanumanthappa, and V. Kavitha, "A survey of machine learning algorithms for big data analytics," in *International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, March 2017, pp. 1–4.
- [15] S. Sotiriadis, N. Bessis, and R. Buyya, "Self managed virtual machine scheduling in cloud systems," *Information Sciences*, 2018.
- [16] L. Sant'ana, D. Carastan-Santos, D. Cordeiro, and R. D. Camargo, "Real-time scheduling policy selection from queue and machine states," in *2019 19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. Los Alamitos, CA, USA: IEEE Computer Society, may 2019, pp. 381–390.
- [17] S. S. Gill, P. Garraghan, V. Stankovski, G. Casale, R. K. Thulasiram, S. K. Ghosh, K. Ramamohanarao, and R. Buyya, "Holistic resource management for sustainable and reliable cloud computing: An innovative solution to global challenge," *Journal of Systems and Software*, 2019.
- [18] M. Xu, C. Q. Wu, A. Hou, and Y. Wang, "Intelligent scheduling for parallel jobs in big data processing systems," in *2019 International Conference on Computing, Networking and Communications (ICNC)*, Feb 2019, pp. 22–28.
- [19] D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, and F. Leisch, *e1071: Misc Functions of the Department of Statistics, Probability Theory Group*, 2019.
- [20] R. Core Team, *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria, 2018. [Online]. Available: <https://www.R-project.org/>
- [21] S. Landset, T. M. Khoshgoftaar, A. N. Richter, and T. Hasanin, "A survey of open source tools for machine learning with big data in the hadoop ecosystem," *Journal of Big Data*, vol. 2, no. 1, p. 24, Nov 2015.
- [22] C. Ferri, J. Hernández-Orallo, and R. Modroui, "An experimental comparison of performance measures for classification," *Pattern Recognition Letters*, vol. 30, no. 1, pp. 27 – 38, 2009.
- [23] M. A. Maniar and A. R. Abhyankar, "Validity index based improvisation in reproducibility of load profiling outcome," *IET Smart Grid*, vol. 2, no. 1, pp. 131–139, 2019.
- [24] W. Zhang, S. Wang, W. Wang, and H. Zhong, "Bench4q: A qos-oriented e-commerce benchmark," in *IEEE 35th Annual Computer Software and Applications Conference*, July 2011, pp. 38–47.
- [25] R. Hood, H. Jin, P. Mehrotra, J. Chang, J. Djomehri, S. Gavali, D. Jespersen, K. Taylor, and R. Biswas, "Performance impact of resource contention in multicore systems," in *2010 IEEE International Symposium on Parallel Distributed Processing (IPDPS)*, April 2010.
- [26] C.-J. Wu and M. Martonosi, "Characterization and dynamic mitigation of intra-application cache interference," in *(IEEE ISPASS) IEEE International Symposium on Performance Analysis of Systems and Software*. IEEE, 2011, pp. 2–11.
- [27] M. Dorier, G. Antoniu, R. Ross, D. Kimpe, and S. Ibrahim, "Calciom: Mitigating i/o interference in hpc systems through cross-application coordination," in *IEEE 28th International Parallel and Distributed Processing Symposium*. IEEE, 2014, pp. 155–164.
- [28] H. Jin, H. Qin, S. Wu, and X. Guo, "Ccap: A cache contention-aware virtual machine placement approach for hpc cloud," *International Journal of Parallel Programming*, vol. 43, no. 3, pp. 403–420, Jun 2015.
- [29] M. M. Alves and L. M. de Assumpção Drummond, "A multivariate and quantitative model for predicting cross-application interference in virtual environments," *Journal of Systems and Software*, pp. 150 – 163, 2017.