Journal of Systems Architecture 116 (2021) 102064

Contents lists available at ScienceDirect

Journal of Systems Architecture

journal homepage: www.elsevier.com/locate/sysarc

ML-driven classification scheme for dynamic interference-aware resource scheduling in cloud infrastructures

Vinícius Meyer *, Dionatrã F. Kirchoff, Matheus L. Da Silva, Cesar A.F. De Rose

School of Technology, Pontifical Catholic University of Rio Grande do Sul (PUCRS), Building 32, Av. Ipiranga, 6681 - Porto Alegre - RS, Brazil

ARTICLE INFO

Keywords:
Interference-aware classification
Cross-application interference
Resource contention
Workload analysis
Machine learning
Cloud computing
Dynamic resource scheduling

ABSTRACT

Computing systems continue to evolve, resulting in increased performance when processing workloads in large data centers due to the virtualization benefits. This technology is the key factor that allows multiple applications to share resources, thereby enhancing the overall hardware utilization of cloud computing environments. However, multiple cloud-services contending for shared resources are susceptible to cross-application interference, which can lead to significant performance degradation and, consequently, an increase in Service Level Agreements violations. Nevertheless, state-of-the-art resource scheduling still relies mainly on resource capacity, adopting heuristics such as bin-packing and overlooking this source of overhead. But in recent years, interference-aware scheduling has gained traction, with the investigation of ways to classify applications regarding their interference levels and the proposal of static interference models and policies for scheduling co-hosted cloud applications. The preliminary results already show a considerable improvement in resource utilization and can be considered as the first steps toward a dynamic scheduling strategy. In this scenario, this paper proposes a machine learning-driven classification scheme for dynamic interference-aware resource scheduling in cloud computing environments. The main goal is to present how a classification approach, that better represents the workload variations, affects resource scheduling. In the first place, we analyze how hardware resources react to different applications with dynamic workloads. Then, we explore distinct interference classification formats and evaluate their efficiency, taking the dynamic nature of cloud workloads into account. Lastly, we present an interference-aware application classifier based on machine learning techniques and compare it with related work, adopting a variety of workload patterns. Preliminary results revealed an improvement in resource utilization efficiency by 27%, on average, when applying our classification approach in cloud infrastructures.

1. Introduction

In recent years, cloud computing has received considerable attention from the scientific community. It is accepted as an ultimate way of managing data utilization and resources, as well as delivering various computing IT services [1]. Many internet-based applications have begun to take advantage of cloud computing due to the promise of unlimited computing resources and the pay-as-you-use model [2,3]. A cloud system can offer this set of capabilities by the benefit of virtualization techniques, executing multiple virtual instances placed on different physical machines, located in data centers [4]. This strategy allows cloud computing providers to use their infrastructure more efficiently, reducing expenses with energy consumption [5]. However, related work [6–8] show that several cloud-services contending for shared resources can generate cross-application interference, which may lead to significant performance degradation and consequently to an increase in Service Level Agreement (SLA) violations.

In order to allow virtualized platforms to deliver SLA guarantees for high user satisfaction, efficient and automatic resource scheduling strategies are essential [9,10]. Resource scheduling is a core function of cloud computing providers and a central component to coordinate all the other platform features to deliver performance-oriented solutions [11]. Typically, in large data centers, resource scheduling is

The code (and data) in this article has been certified as Reproducible by Code Ocean: (https://codeocean.com/). More information on the Reproducibility Badge Initiative is available at https://www.elsevier.com/physical-sciences-and-engineering/computer-science/journals.

* Corresponding author.
E-mail addresses: vinicius.meyer@edu.pucrs.br (V. Meyer), dionatra.kirchoff@edu.pucrs.br (D.F. Kirchoff), matheus.lyra@edu.pucrs.br (M.L. Da Silva), cesar.derose@pucrs.br (C.A.F. De Rose).

https://doi.org/10.1016/j.sysarc.2021.102064
Received 15 May 2020; Received in revised form 19 January 2021; Accepted 15 February 2021
Available online 20 February 2021

accomplished through heuristics, such as bin-packing, which considers only resource capacity aspects, overlooking their source of overhead [12,13].

Looking for alternatives, recent related work [14] has proposed static models and scheduling policies based on interference generated by co-allocated applications. That study suggests an attraction/repulsion method built upon the workload profile of each application, getting around the traditional concept of just observing resource utilization and capacity. In this case, web applications are investigated since they are a category that presents workload variations at runtime. Besides, they have an unpredictable intensity variation of resource utilization due to different user request patterns and periodicity [15].

Performance interference among web applications in multi-tenant systems is known to adversely impact the Quality of Service (QoS) properties of applications [16]. Dynamic service demands and workload profiles further raise the challenges for cloud service providers in managing resources on-demand to satisfy SLAs while minimizing operational costs [17]. Any solution that addresses these challenges requires an approach that should account for the workload variability and performance interference [18]. Therefore, evolving from a static to a dynamic interference-aware scheduling strategy brings some questions up, given the inherently dynamic nature of the process, such as: How to classify dynamic workloads based on resource interference in real-time? When to execute the classification?

In order to start answering these questions, this paper proposes an interference-aware classification scheme that aims at enabling better utilization of available resources through the execution of multiple workloads on the minimum number of physical machines, and, consequently, minimizing SLA violations. Usually, any scheduling approach must consist of at least two parts: a classification scheme for identifying which applications should and should not be scheduled together, as well as the scheduling policy that makes the decisions based on this information [19]. Since the classification component plays a pivotal role in dynamic resource scheduling in cloud computing environments, in this work, we focus on the analysis of a classification scheme and its efficiency. Firstly, we perform an initial analysis of how applications with dynamic workloads behave under distinct circumstances, observing each resource and its contrast to response time aspects. Then, we describe how an interference-aware classification scheme, that better represents the workload variations, handles computational resources and its effect on resource scheduling. Finally, we present an interference-aware application classifier based on the combination of two well-known machine learning techniques, validating its model through some quality measures, and then comparing the classifier with related work.

The rest of this document is organized as follows: Section 2 discusses related work and background material. Section 3 presents an analysis of interference from dynamic workloads. Section 4 demonstrates different interference classification formats and their impact on resource scheduling. Section 5 describes an interference-aware application classifier, its functionalities and capabilities. Section 6 introduces related work in the literature. Finally, Section 7 depicts conclusions and future directions.

2. Background and state-of-the-art

This section outlines the state-of-the-art concepts intrinsic to the work. Firstly, we characterize applications with dynamic workloads. Then, we define interference and its impact on performance. After, we introduce the interference-aware scheduling concept. Lastly, we explain the machine learning techniques adopted in this study.

2.1. Applications with dynamic workloads

In cloud computing, applications may have different workload patterns and QoS requirements. For example, non-interactive batches require completion time, while transactional web applications are concerned with throughput guarantees. Different application workloads demand a diverse type and amount of resources. In particular, batch-jobs tend to be relatively stable, while latency-sensitive tends to be highly unpredictable and bursty in nature [20]. Besides, latency-sensitive applications can include short latency-critical user-facing tasks, such as responding to web search requests. Also, this class of workloads is characterized by short deadlines in the order of tens of milliseconds [21].

Multi-tenant web systems can efficiently allocate resources within and among data centers according to time-varying demand [15]. Their workload is not deferrable, and this means that every time a request is received, the response should be generated immediately afterward. Consequently, such applications must make real-time scheduling of the load, which does not delay the current requests [22]. This application category presents workload variations at runtime and an unpredictable intensity variation of resource utilization due to the user's different request patterns and periodicity [15]. Therefore, web applications are ideal candidates to evaluate interference effects suffered by dynamic workloads and will be considered as target applications in this work.

Garg et al. [20] propose an admission control and scheduling mechanism to ensure the meeting of users' QoS requirements, as specified in SLAs. The authors claim that it is important to be aware of different types of SLAs and the mix of workloads for better resource provisioning and utilization of data centers. Results show substantial improvement in reducing SLA violations. Sampaio et al. [23] address the resource allocation issues running different types of application workloads, such as CPU- and network-intensive applications. After conducting several experiments with synthetic workloads, results indicate that the author's strategy is able to fulfill contracted SLAs of real-world environments while reducing energy costs.

2.2. Performance interference

Nowadays, due to resource sharing techniques, physical machines host several applications. Resource sharing methods, such as virtualization and containerization, provide several approaches to mitigate resource contention problems related to co-hosted applications. Such technologies are the main drivers of high resource utilization in modern data centers. However, the intensive use of a source, considering multiple services, certainly will face resource contention problems. This problem is known as performance interference, and it may lead to severe performance degradation [14,24].

Resource consolidation may lead to severe performance degradation. The performance degradation caused by virtual machines running on the same computational environment is known as virtual machine interference. Handling higher virtual machine interference may result in a higher consolidation, while strict low interference requirements may demand more resources. 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%. In memory and disk I/O intensive applications, the performance degradation was 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.

Besides the environments mentioned above, performance interference also affects container-based environments. 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 environments usually run multiple user-applications concurrently, handling several requests that eventually compete 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. [25] 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%.

2.3. Interference-aware scheduling

In cloud computing ecosystems, consolidating multiple user applications onto multi-core servers generates interference between co-hosted applications, which impacts application performance. To minimize interference effects and improve application performance, a common solution is to utilize schedulers that consider interference issues [26].

Chiang and Huang [27] present TRACON, a Task and Resource Allocation CONtrol framework that mitigates the interference effects from concurrent data-intensive applications and improves the overall system performance. TRACON utilizes machine learning-based techniques to perform an interference model prediction that infers application performance from resource consumption observed from different VMs and an interference-aware scheduler that is designed to utilize the model for effective resource management. Evaluation results present an improvement of 50% on application at runtime and 80% on I/O throughput for data-intensive applications in virtualized data centers.

Kansal and Ghaffarkhah [16], Delimitrou and Kozyrakis [28] and Shekhar et al. [18], concerned about meeting QoS requirements while improving the system efficiency, have built online interference-aware schedulers. They apply predictive models that observe workload metrics and performance interference interactions to perform resource management. Experimental evaluations show system utilization improvement by up to 35% [16]; QoS guarantee rates of 52% [28]; and tail latency reduction by up to 39% [18].

Zhu and Tung [29] and Bu et al. [30] 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. [31] and Wang et al. [32] developed 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. [12] 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.

2.4. Machine learning algorithms

Some problems are solved from algorithms specifying step-by-step how they can be solved. However, it is not an easy task to write a program to mimetize human abilities, such as face or voice recognition, which demands experiences and pattern recognition. Therefore, machine learning is a technique that learns from data, patterns analysis, and approximation functions to extract knowledge based on the input data [33]. Machine learning techniques are mainly grouped into three categories: (i) reinforcement learning, (ii) supervised, and (iii) unsupervised. Reinforcement learning allows a machine to learn from the feedback received through interactions with an external environment. Unsupervised machine learning is applied to conclude from a given dataset consisting of input from a data. Supervised machine learning techniques estimate the relationship between input attributes and a target attribute. Thus, they can be classified into two main categories: classification and regression. On the regression, the output variable takes continuous values; on the other hand, while in classification, it split the data into different classes [34]. In this study, two machine learning algorithms have been employed: SVM for classification and K-Means for clustering.

2.4.1. SVM

Support Vector Machine (SVM) is a supervised technique based on statistical learning theory. Support vectors are the data points nearest to the hyperplane. Hence, the main idea is finding a hyperplane that best divides a dataset with the minimum cost into two classes in a dimensional space. [35] Sotiriadis et al. [36] minimized performance degradation in cloud computing, introducing a virtual machine scheduling algorithm. They apply SVM to classify resource usage. As a result, performance degradation has been minimized by 19%, and CPU real-time has been maximized by 2%. Sant'Ana et al. [37] 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%.

2.4.2. K-means

K-Means is an unsupervised technique that attempts to split a given dataset into a fixed number of clusters. In this method, 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 employed to produce the final clustering. Gill et al. [38] propose a resource scheduling technique for holistic management of cloud computing resources. This method uses K-Means for clustering the workloads for execution on a different set of resources. Results indicate that the 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. [39] 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.

3. Interference of dynamic workloads

Uncontrolled access to shared resources can cause performance variations, leading applications to perform unsteadily or even fail. The friction caused by the competition to access cache, memory, disk, or internal busses is called resource contention [40]. I/O contention, for instance, occurs when multiple tasks compete for a portion of disk bandwidth in a context where the demand is higher than the available resources. The constant expansion of data centers has raised a concern regarding resource contention issues, where performance aspects are crucial, and SLA cannot be violated, which is the case of cloud computing infrastructures [8]. In order to understand interference effects under dynamic workloads in a more appropriate way, first, we need to understand the basics of how does each resource behaves and is handled by internal devices while the workload varies. Furthermore, we may need to understand what problems they might cause when contention-related issues are applied to all those types of resources. In every aspect, each of the main resources such as cache, CPU, memory, disk, and network, are those that suffer the most by consolidated applications that may share the same infrastructure [6].

This section aims to explain how the interference is profiled and how applications with dynamic workloads act under different circumstances. Initially, we introduce the interference profiler used in this work. After, we present a resource interference analysis, and lastly, its impact on response time.

3.1. Profiler and setup environment

To characterize the interference generated by each application, we used a tool called IntP [8]. It profiles running applications using low-level kernel instrumentation, returning the interference levels generated on each resource subsystem. Although IntP can be used to provide metrics in realtime because of its low overhead, in this work we used it only as an offline profiler to generate a dynamic interference classification of the applications that will be used later in the dynamic scheduling. It is structured in modules that are responsible for each type of access on a specific resource at the infrastructure level, and outcomes the percentage of hardware resources utilization, per application, in an isolated fashion. This isolated measurement provides analytical information to the system to determine how much an application interferes with each other. The higher the metric is, the more interference the application is profiled generates. More specifically, the tool returns the percentage of interference of the 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;

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

3.2. Resource analysis

To perform an in-depth analysis of interference generated on different hardware resources from applications with dynamic workloads, Node-Tiers[1] has been adopted. This tool is a multi-tier benchmark that allows fine-grained personalization of resource utilization. Node-Tiers stresses the computer system in various selectable ways and was designed to exercise various physical subsystems of a computer through web requests. This tool explores the web applications concept (client–server) and allows the creation of workload variations. The server-side was performed over a server (presented in Section 3.1). While the client-side was configured on a different computer. Both pieces of equipment were connected through a Gigabit Ethernet Network. The goal is to stress a server in many ways (distinct resources), through latency-sensitive applications, increasing the request arrival rate, and observing interference effects over the changes in workload behavior.

Firstly, we have chosen Node-Tiers algorithms that stress more a given resource. After, an increasing workload has been created, varying from 0 to 300 requests per second, within 300 s. Each algorithm was executed in two ways: in isolation and two applications' instances co-hosted, labeled here as parallel. Fig. 1 presents all experiments' results, depicting how each resource is affected by interference over time. Below, a detailed analysis is discussed.

3.2.1. Cache

Last Level Cache (LLC) memory is a hardware device created to minimize the performance gap between the processing cores and the main memory [41]. When two or more processes are assigned to the same CPU node, threads occasionally share on-chip memory space, and it may lead to resource contention [8]. At the isolated execution, it is possible to observe that the resource that suffers the highest interference is the cache, with an increasing but non-linear behavior. The next most affected resources are memory and CPU, which pursue a linear trend. In parallel execution, cache interference increases significantly, and some peaks in cache-miss occurred at the end of the experiment. Although the interference suffered by the memory is following a linear trend, this resource interference increases more than twice in parallel compared to isolated execution. The same happens with the CPU metric.

3.2.2. CPU

Multiple co-hosted applications running might outstrip the available amount of CPU cycles, when this happens it is called CPU contention [8, 42]. In our experiment, even though the target resource to be stressed is the CPU, the cache is the one that suffers the most from interference. The CPU follows a linear trend, both in isolated and parallel execution. In this case, the CPU, on average, duplicates over time, generating the highest proportionality among all experiments. The memory rates seem to double from isolated to parallel tests as well.

3.2.3. Memory

Memory contention happens when the memory requirements for the active processes exceed the available system physical memory, causing the system to run out of memory while dramatically decreasing the system performance. To overcome this problem, the operating system (OS) present two possible mechanisms: (i) System Paging, when the OS starts to move fractions of active processes to the disk and tries to recover physical memory and reestablish stability; and (ii) System Swapping, when the OS starts to swap an entire process to the disk to reclaim memory, causing tremendous disk overhead [43]. In the isolated execution, the memory follows a linear tendency until a given amount of requests. When requests overtake such a quantity, the CPU increases considerably, reaching interference rates over 90%. This abrupt CPU growth happens when the request arrival rate reaches more than 300 requests per second, approximately. Since workloads are co-hosted in parallel execution, that request rate is reached earlier, close to 150 s, creating a significant resource usage increment. So, it is possible to note that not only resources are widely consumed, but the experiment that should have finished within 300 s, finished close to 310 s, generating performance degradation.

3.2.4. Disk

Disk throughput can be seen as the most volatile performance metric in a system, because it is architecture-driven and might be affected by external components, such as virtual memory, buses, and I/O controllers [44]. For isolated execution, the increase in the disk is smoothed, following a linear trend. In parallel execution, the interference that co-allocated applications generate follows an exponential trend. Since the arrival request rate is doubling every second, it is possible to observe that interference generated by the block storage contention is significantly amplified.

3.2.5. Network

Network contention appears when processes send messages that travel over the same network interface card (NIC), passing through internal buffers concurrently, thereby increasing job communication time and degrading performance [45]. In isolated and parallel execution, network back-pressure grows linearly. Comparing both executions, it is possible to note that even if the load doubles in parallel execution, network interference does not double proportionally. Besides, in parallel execution, the use of cache is intensified, generating some cache-miss incidences and presenting a strong relationship between network and cache resources.

[1] https://github.com/uillianluiz/node-tiers

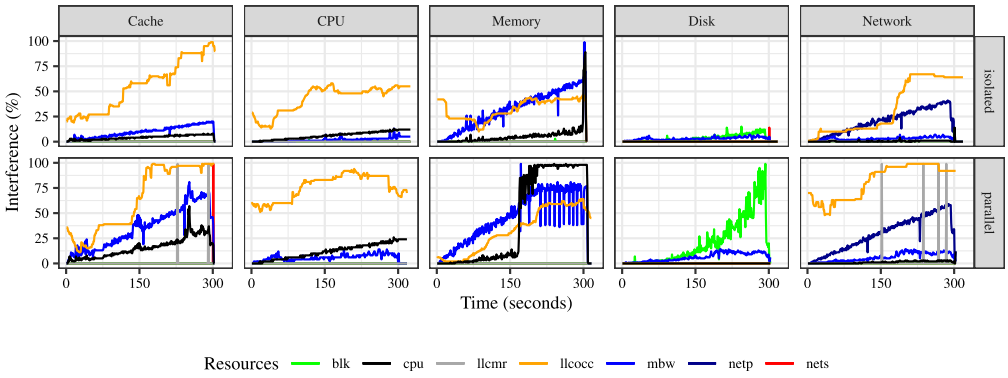

Fig. 1. Isolated and Parallel executions from Node-Tiers algorithms that stress the more cache, CPU, memory, disk, and network resources.

3.2.6. General remarks

In general, we can conclude that each application has a specific resource usage behavior, creating different interference rates. Besides, when observing specific resources, we highlight that applications can generate interference indexes according to the workload variation, proportionally to the load changes, which are the cache and CPU examples. On the other hand, the opposite happens as well, when the interference generated does not follow the workload rise proportionality, that are the cases of memory, disk, and network. Also, we state that these analysis results could change if they were executed over different hardware, strongly depending on their characteristics or capabilities.

3.3. Impact on response time

Left unmanaged, the competition for machine resources can lead to severe response-time degradation and unmet SLAs [46]. Latency-sensitive does not regularly imply hard real-time applications but also applications that have soft bounds on response times beyond which users will find the application behavior unacceptable. For instance, users expect a web search to complete within a specific amount of time [18]. So, delays in latency-sensitive applications runtime do not always represent properly their performance degradation or demonstrate SLA violations. Therefore, to evaluate such applications deterioration accurately, we have analyzed the response time degradation through a set of experiments. With Node-Tiers, we have combined all algorithms, mentioned in the previous subsection, in parallel executions and captured their response times. To better explain, the application that uses more the cache was combined with the one that uses more the CPU, memory, disk, and network. This experiment has been performed with all algorithms, matching all of them. All executions take the same workload and time interval: 0 to 300 requests per second within 300 s. Fig. 2 depicts all the collected response times from all tests.

It is possible to notice that each application has its specific increment in response time and none of them have a linear growth trend. For example, in the CPU–network execution, the Network application starts to increase its response time before the CPU. In the CPU–memory case, practically memory does not change its response time behavior while CPU does.

Another interesting remark is that all executions with the same application, in the diagonal line (i.e. cache–cache, CPU–CPU, and so on), present the same behavior and start to increase their response times practically at the same time. This happens because the same resource is being used, causing approximately the same response time degradation.

It is worth mentioning that although each application has its characteristics and different behavior, when co-hosted with another one, the interference generated between them tends to affect proportionally their response times, on average. This means that response time degradation is proportional to each resource.

Table 1
Interference intervals and their respective level labels, introduced by Ludwig et al. [14].

Interval	Label
0%	Absent
1%–20%	Low
21%–50%	Moderate
51%–100%	High

4. Workload interference classification

Resource scheduling can be defined as the ability of cloud infrastructures to dynamically change the amount of resources allocated to a running application. Hardware resources should be allocated according to the change of workload, allowing the management of resources to preserve the Quality of Service requirements at reduced cost [47]. In the previous section, it has been stated that workload variations can affect differently the behavior of applications, not only in resource usage but also in response time aspects. Each application execution might have a different hardware subsystem comportment, strongly depending on the workload variability. Therefore, an interference classification scheme that perceives the changes of application behavior over time becomes essential to perform interference-aware scheduling strategies in cloud computing environments.

The most challenging part of the problem is to find a classification scheme that can accurately determine how cross-application interference affects each resource when they are being shared over time [19]. To this end, this section presents a preliminary evaluation of how different interference classification techniques impact interference overhead and resource utilization. First, we discuss the impact of workload variation on the classification phase of these techniques. Then, we evaluate their efficacy in reducing the overhead and improve resource utilization in such a scenario.

4.1. Classification analysis

To evaluate scheduling policies alternatives, Ludwig et al. [14] have created a classification method that explores interference at levels. Such method analyzes the interference suffered by machine resources (CPU, memory, disk, network, and cache) over the entire application execution. The authors' technique categorizes such resources with their respective interference level labels, according to Table 1.

To introduce an example, we have run a QoS-oriented e-commerce benchmark called Bench4Q [48]. This application has features to deduce a controllable and flexible representation of complex session-based workloads and to simulate authentic customer behavior. First, we created an increasing workload, starting with a low load and gradually going to a high load, and profiled it with IntP. Bench4Q emulates active e-commerce users through entities called Emulated Browsers (EBs), so

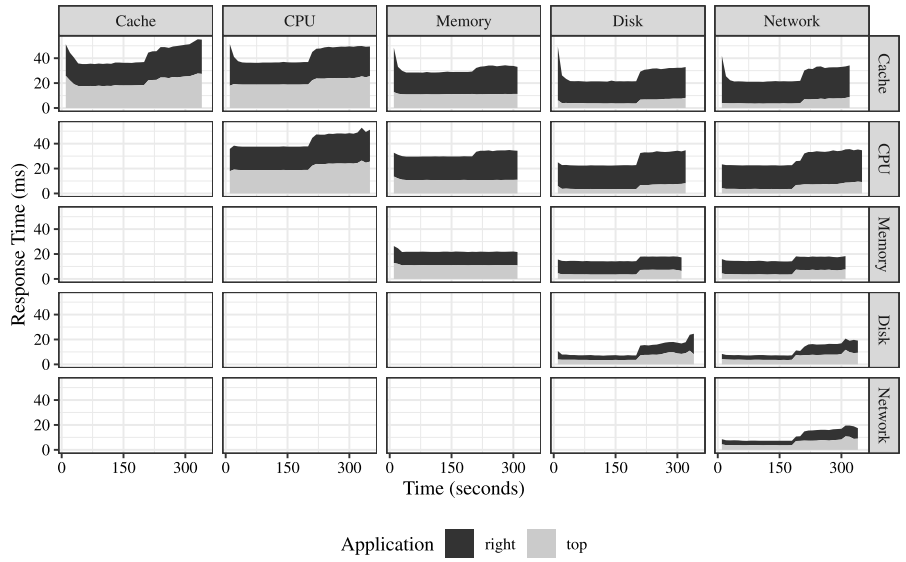

Fig. 2. Response time collected combining cache, CPU, memory, disk, and network algorithms from Node-Tiers.

the load started with 10 simultaneous EBs and every minute we added 10 more, ending the execution with 120 EBs (720 seconds). Fig. 3 shows interference suffered by each resource in this experiment. The top chart presents the static classification method proposed by [14], which analyzes the interference levels over the entire application life process and assigns just one label per profiled resource based on mean values. In this work, we refer to this classification format as Unique. To evaluate how well this technique deals with workload variations and its impact on the classification, we redid the same experiment segmenting the trace in four parts and applying the same static classification technique to each part. Results are shown at the bottom chart of the same figure and we refer to it as Segmented.

It is possible to notice that there are resources that do not change their labels, for instance, memory, cache, and network. Since they keep their interference metrics at the same level, on average, with no expressive variation, their labels are maintained. On the other hand, also some resources do change their labels, which are the CPU and disk cases. The disk has a smooth decrease in its behavior, moving from low to absent label, at the execution halfway. Besides, CPU has the biggest behavior change, starting with low, going to moderate levels, and ending with a high interference level.

While there exist applications with expressive variations in the interference metrics over a given period, we have noticed there exist those that do not present it. To better explain, we did the same experiment with a benchmark developed to evaluate database performance for workloads, similar to those of Facebook's production, named LinkBench.[2] This application can be configured to simulate a variety of workloads, and plugins can be written for benchmarking additional database systems. Again, an increasing workload has been created. In the LinkBench benchmark, it is possible to set the number of requests (operations) and the number of requesters (threads). The number of requests was configured into 1000 (fixed), and the number of requests varied from 10 to 50 (by 10 to 10). The entire execution was profiled with IntP. Also, a single classification was performed on top of the entire application execution. After, the execution was partitioned into four parts as well, and each part was classified again. Results are presented in Fig. 4.

The top chart (Unique) shows that only disk and cache suffer low rates of interference, in general. By classifying the application in four parts, in the bottom plot (Segmented), it is possible to notice that the overall interference generated in each resource keeps the same labels. This means that the interference metrics do not change significantly to modify their labels. Concluding that: (i) there are cases where interference levels do not change, even when their workload does; and (ii) in these cases, the Ludwig et al. [14] classification method is well applied since there is no representative variation in the interference metrics.

Table 2
Performance degradation generated by resource interference, introduced by Ludwig et al. [14].

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

Summing up, while Bench4Q execution generates a significant variation in terms of interference effects, LinkBench does not. This highlights that, due to their dynamic workload nature, each application should be handled differently.

4.2. Impact on interference overhead

Aiming to evaluate interference changeability and scheduling aspects over time, by profiling the total interference generated of running an application, a tool called CIAPA[3] has been adopted. This is a scheduling analysis tool that uses an interference overhead function, 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 Eq. (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 in which they are set, the interference overhead index value varies according to Table 2.

CIAPA tries to minimize the total interference overhead by testing all possible combinations of applications per host. Therefore, the result

[2] http://github.com/facebookarchive/linkbench

[3] https://uillianluiz.github.io/ciapa

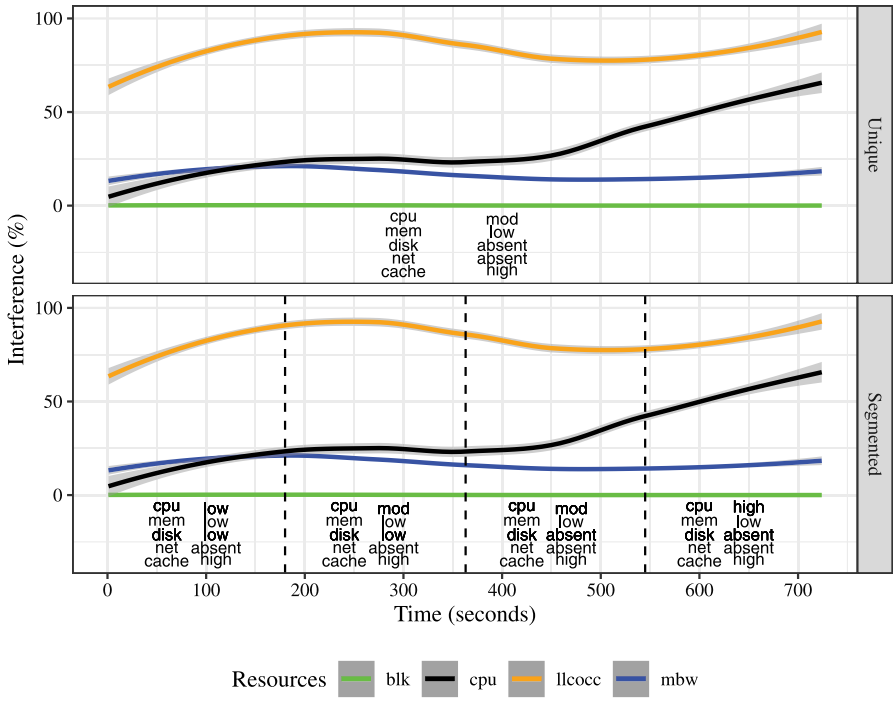

Fig. 3. Unique (top) and Segmented (bottom) Bench4Q static interference classification. To facilitate the visualization, a Loess function was applied to smooth short-term variations in each resource. Resources labels that changed are shown in bold in the bottom plot. IntP metrics that do not suffer any interference in these experiments were not depicted.

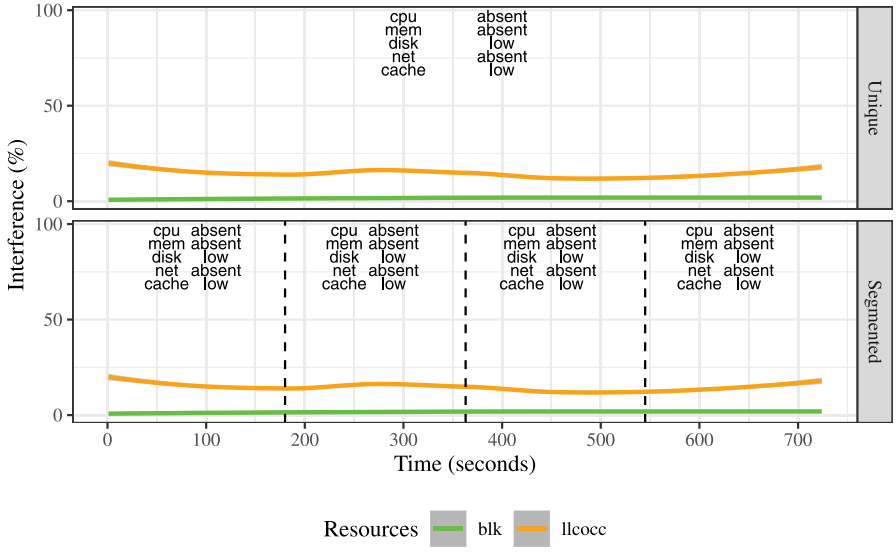

Fig. 4. Unique (top) and Segmented (bottom) LinkBench static interference classification. To facitlate the visualization, a Loess function was applied to smooth short-term variations in each resource. IntP metrics that do not suffer any interference in these experiments were not depicted.

is finally given by the multiplication of the cost of each resource, which is calculated by using the function seen in Eq. (2).

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

To execute applications with dynamic workloads, we have elected three different applications that can execute workload variation. The first and second are the already mentioned Bench4Q and LinkBench. The third one is a decision support benchmark called TPC-H.[4] This application 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 behavior of the application in terms of resource usage and performance. In order to variate their comportment, 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. [15] 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 load variations. *Decreasing* is the opposite of *Increasing*, it starts with a high load and gradually goes to a low load. Finally, *Constant* keeps always the same workload.

[4] http://www.tpc.org/tpch/

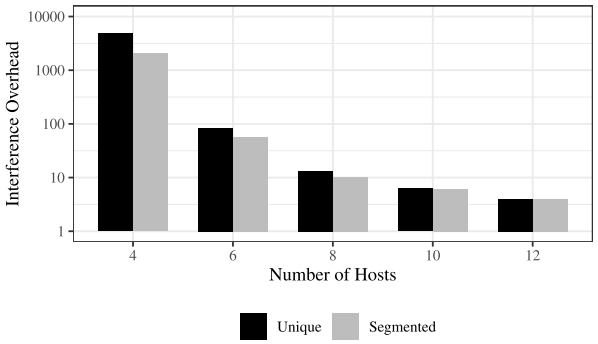

Fig. 5. Comparison of interference overhead.

To evaluate interference variations, each application was submitted to the 4 workload patterns, resulting in 12 different examples. These examples have been tested over different numbers of hosts, as follows: 4, 6, 8, 10, and 12. After, we have applied two classification formats on each workload: (i) Unique and (ii) Segmented. Where, Unique represents just a single classification over the entire application execution, while Segmented divides each execution into four parts and runs the classification on each one. Since, to perform the Segmented format, the application was divided into four parts and each part outcomes a set of interference level labels, the outcomes from the Unique format were multiplied by four in order to create a proportionality between both formats.

Classification outcomes were inserted into CIAPA, and the results are presented in Fig. 5. As mentioned before, CIAPA uses an interference overhead function that represents the total interference generated into the system. The lower this result, the better the hardware is being used. This means that lower indexes present better hardware efficiency so that reducing response time, makespan, and so on.

In general, the largest difference has occurred with the smallest number of hosts. These were the cases that happened more cross-application interference. Resulting in greater performance degradation among co-located applications. With the growth of hosts number, the difference between interference overhead incidence decreases. Therefore, resource concurrency among co-hosted applications tends to decrease, as well. Only with the largest number of hosts, both classification methods, 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 result has been led to the minimum.

It is interesting to note that in all experiments, performed in Fig. 5, the Segmented format reaches lower interference overhead indexes than the Unique one. The results demonstrate that Segmented format improved the hardware utilization efficiency by 22%, on average, reducing resource consumption and also performance degradation at the application level. This highlights that a workload-aware fine-grained classification can reduce interference overhead while a single one can lead to less efficient scheduling decisions. Besides, evaluating cross-application interference at time intervals tends to improve the performance of applications while preserving the Quality of Service requirements.

From now on, in this paper, the *Classification Method* refers to the method of how does each resource receives its interference level within a given execution period, while *Classification Format* indicates how many time slices the Classification Method is applied in: Unique format for a single interference classification over the entire application execution, and Segmented for multiple classifications.

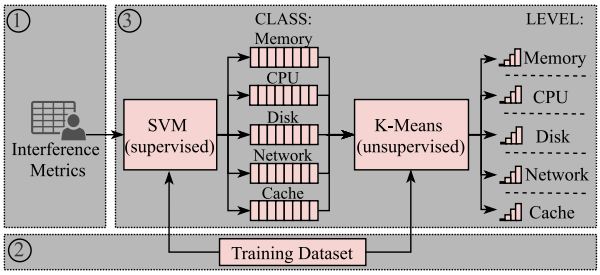

Fig. 6. Classifier architecture overview: Component 1 represents the collecting of interference metrics; Component 2 depicts the training dataset assisting machine learning algorithms; Component 3 illustrates the classification process and its outcomes [49].

5. ML-driven interference-aware application classifier

In the previous section, we did some experiments under high workload variations running a static interference classification only once (Unique) and several times along the execution (Segmented). Comparison results showed that the Segmented format reaches better resource utilization efficiency than the Unique one. However, the interference levels' thresholds (Table 1) of the applied static classification method [14] were empirically defined. Although this still resulted in applications' scheduling 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 face problems when dealing with different types of applications that have dynamic workload patterns.

To tackle this issue, we created an interference-aware application classifier based on the combination of two well-known machine learning techniques: Support Vector Machines (SVM) and K-Means. After trained, the proposed classifier receives monitored metrics from applications and dynamically defines their interference levels thresholds for each resource. This application classifier was first introduced in [49] in its static variant, and here we improve this work by introducing a dynamic version, where we apply the classifier several times during the execution of an application to better react to workload variations and possible changes in its interference levels.

This section details the classifier functionality and presents an evaluation of its utilization, performing different experiments. First, we explain the overall functionality of how does the proposed classifier work, including its dependencies and capabilities. Then, we present an evaluation of the dataset and model validation. Lastly, we compare its efficiency with related work.

5.1. Classifier design

Two machine learning algorithms work together to implement the proposed classifier: SVM for classification and K-Means for clustering. Initially, SVM receives interference metrics from the target application collected each second by IntP, and those metrics are classified and stored into resource queues for their respective classes: memory, CPU, disk, network, and cache. Subsequently, K-Means quantifies values for each queue and returns their interference level for a specific period. Both machine learning algorithms use a training dataset, previously defined, to assist their decisions. Fig. 6 illustrates an overview of how the classifier works. More details about the classification method are presented in the next subsections.

5.1.1. Interference profiling

To characterize the interference generated by each application we used IntP. The general idea is to profile applications at runtime, returning an interference for each resource used during execution: memory (mbw), CPU (cpu), disk (blk), network (netp and nets), and cache (llcocc and llcmr). IntP returns these metrics every second, based on the workload variability (higher values correspond to a higher interference overhead).

5.1.2. Training dataset

To take advantage of selected machine learning techniques, it is essential to have as much quantity of data as possible to use as input in order to train the models. However, no available datasets were found in the literature with cross-application interference traces. To tackle this issue, Node-Tiers has been used. Since this tool is able to stress the computer system in many ways, through web requests, we could generate a diversified interference dataset by performing various algorithms.

To maintain a data history from each interference class, we had to stress the main resource classes and store their interference metrics. To better understand, let us take an example: To collect CPU interference metrics, Node-Tiers has been set with *cpu* parameter, which means only the CPU was stressed. To collect the cache class, *cache* parameter has been set, and so on. We have produced five major interference classes: *memory*, *CPU*, *disk*, *network* and *cache*. More precisely, 10,000 samples have been collected from each class of interference, resulting in a dataset with 50,000 samples.

5.1.3. Classification process

The proposed ML-based interference classifier dynamically defines thresholds and assigns interference levels for each resource used by the monitored applications for a particular time slice, without the need for user intervention.

The target application is monitored with the IntP tool, every second, generating data from 7 resource metrics (described in Section 3.1). This data is passed to SVM as input data to be classified. SVM is a supervised technique, so it uses labeled data from a training dataset to label the new data. SVM takes this tuple of 7 interference values and categorizes it into one of the 5 resource classes: memory, CPU, disk, network, and cache. The idea here is to select the class that best represents the interference generated by this application in this second and store the interference value(s) in its respective queue. Since this is done at runtime for all applications that are being executed, it is a way to reduce overhead, selecting the most representative values for each class. After a system defined time interval, the values stored in these resource queues become K-Means input data, so that the corresponding interference level for each resource is assigned. This two-step classification process is repeated until the end of the execution, characterizing the dynamicity of our approach, where interference levels are reevaluated regularly so that we are able to better react to changes in the workload.

Inspired by [14], we used four possible levels: absent, low, moderate, and high. But in our work, these thresholds are not empirically predefined, but variable for each resource and automatically defined in the K-Means training phase. When there is no interference incidence from a class (no data in the respective queue for the period), the classifier interprets it as Absent.

K-Means, previously trained, determines the interference levels of each resource class. K represents classes division: low, moderate, and high. Therefore, its value has been set to 3 ($k = 3$). At the beginning of the classification process, SVM trains its model and K-Means finds its centroids. Since both techniques are supported by an already created dataset, the SVM model and K-Means centroids are the same until retrained.

For example, we may have an IntP tuple for a running application (i.e. [0%, 0%, 3%, 15%, 5%, 10%, 80%]) as input in a particular second classified by SVM as "CPU". This output is buffered in the respective class queue, in this case CPU. This classification phase is repeated for the duration of a time interval, and then these class queues become K-Means input to determine an application interference level for each class within the monitored interval. These queues are only used to buffer the SVM output for each class since SVM runs each second and K-Means runs for each time interval. The goal here is to reduce overhead since this is done at runtime.

The classifier was implemented using the free statistical software tool R.[5] To execute SVM and K-Means algorithms we have elected *e1071* [50] and *stats* [51] R packages, respectively. Basically, SVMs can only solve binary classification problems. To allow for multi-class classification, `libsvm` performs the "one-against-one" technique by fitting all binary subclassifiers and finding the correct class by a voting mechanism [50]. Even though we have chosen R in this work, the model design is not limited to this specific tool, other software or libraries, such as Keras[6] for Python or Weka[7] for Java, could also potentially be utilized. One factor in choosing (or dismissing) a machine learning platform is its coverage of existing algorithms [52]. R provides flexibility for implementing several types of model architectures.

Many machine learning techniques are available in the literature, and depending on the chosen one, there are different sets of parameters to be configured. 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*[8] 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 application classifier analyzes workload behavior through interference metrics, this approach can be applied with other interference metrics and monitoring tools (like PAPI[9]). All files, including source codes and results, are available at GitHub[10] and Code Ocean [53].

To better understand the classifier functionality, Fig. 7 presents the execution of Bench4Q under an oscillating workload. Each class of interference is demonstrated separately, together with its classification level for the elapsed time. Overall, CPU and cache are more stressed than memory, disk, and network. It is worth noting that disk and network resources have undergone low interference rates, on average less than 2% and 5%, respectively. In this case, network classification is categorized as absent while the disk is labeled as low. Although memory values seem not particularly elevated (less than 25% on average), our classifier labeled this resource class as high, based on the K-Means threshold setting.

5.2. Classifier evaluation

In this section, we analyze the training dataset and validate the proposed ML model presenting its quality metrics.

5.2.1. Dataset analysis

In reference to data analysis, machine learning techniques process the dataset and produce a set of descriptive statistics on the included features. In addition to a handful of metrics, these techniques support statistics on configurable data slices and cross-feature statistics such as the correlation between them. The correlation between a component and a variable estimates the information they share. The variables can be plotted as points in the component space using their correlation as coordinates [54].

Fig. 8 presents the Circle of Correlations. This image shows the correlations between all interference metrics collected to develop the training dataset and the principal components (PC) are shown via coordinates. The relationships between all variables can be interpreted as follows:

[5] https://www.r-project.org/
[6] https://keras.io/
[7] https://www.cs.waikato.ac.nz/ml/weka/index.html
[8] https://cran.r-project.org/web/packages/caret/index.html
[9] https://icl.utk.edu/papi/
[10] https://github.com/ViniciusMeyer/interference-classifier

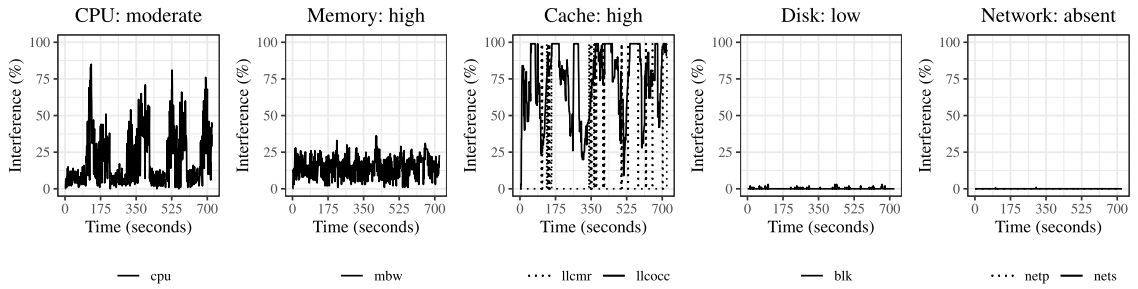

Fig. 7. Bench4Q execution under an oscillating workload.

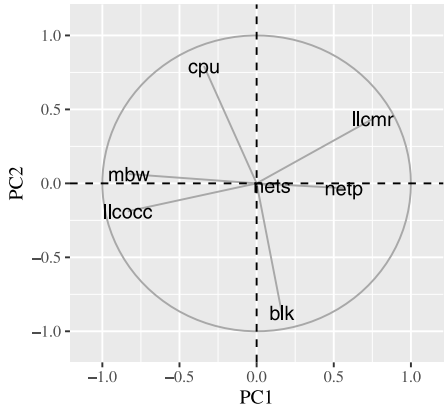

Fig. 8. Correlations circle of dataset interference classes.

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

By looking at this image, observing all features, we can get important insights into the shape of the dataset. It is worth noting that there is a strong positive correlation between some interference classes, such as *llcocc* and *mbw*. On the other hand, there are those that present a negative correlation, such as *cpu* and *blk*. This means that: (i) while the cache is used, memory bandwidth is used as well; and (ii) while CPU is consumed, the disk is practically not required. This information is essential for the clustering phase (K-Means) since it uses those data to train and find the optimal centroids arrangement (interference interval levels). It is noteworthy that information comes from the training dataset, and if we change it, the correlation between resources probably will have different behavior (directions), strongly depending on the data.

5.2.2. Model validation

Usually, when a model is created with the support of machine learning techniques, a validation step is performed to find out if it has a good quality rate. We validate that a model is safe to serve when a simple premise is reached: the quality measures have to achieve reasonable rates. For this purpose, we have elected two classification quality measures [55]: (i) Accuracy is the most common measure to evaluate a classification process. 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. The SVM algorithm was evaluated by repeating a 5-fold stratified cross-validation 10 times, with different randomly selected partitions. This mechanism finds the model with the highest validation score.

Table 3
Quality measures of machine learning techniques. (– not applicable)

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

For clustering, we have defined Rand Index [56] (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 divisions yields a Rand Index of essentially zero. Complete association yields an index of 1.0. From a mathematical standpoint, this index is related to 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. All mentioned metrics are presented in Table 3.

In all experiments, the quality metrics presented acceptable rates. This means that both machine learning techniques used in the classifier, induce a good training quality.

5.3. Comparison with state-of-the-art

To evaluate the proposed dynamic ML classifier, we compared it to two static classification approaches (**Ludwig et al.**), that uses customized classification intervals, and a variation of it that uses proportional intervals (**Proportional**), to verify our claim that we would cope better with applications that have dynamic workloads. We also compare it to the state-of-the-art in round-robin scheduling as a baseline (**Even**). More details about these techniques are presented below:

- **Even** implements the *EvenScheduler* algorithm, which is the Apache Storm[11] default scheduler. This algorithm distributes computation tasks across nodes in a Round-Robin manner [57]. When tasks are scheduled, this approach counts all available slots on each node and places application instances to be scheduled one at a time to each node while keeping the order of nodes constant. We have decided to use this method because Apache Storm is a well-known framework that processes real-time data, like cloud multi-tenant systems, which are the target applications in this work. Besides, we consider *Even* as a baseline since it is the less optimized approach.

[11] https://storm.apache.org/

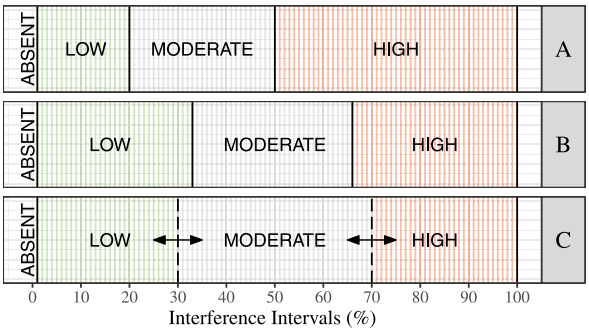

Fig. 9. Interference levels used in each classification method and their respective intervals. Ludwig et al. (A) and Proportional (B) use static thresholds for determining their interference classes while in our classifier (C), they are variable, automatically defined without user intervention.

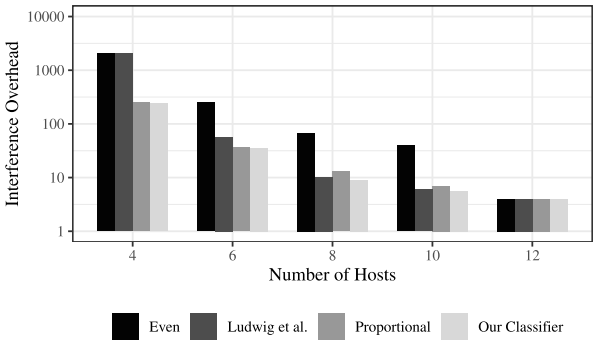

Fig. 10. Comparison of interference overhead with state-of-the-art.

- **Ludwig et al.** [14] evaluate the profile of the application workloads and uses an interference classification at levels. This approach was introduced with details in Section 4.1 and was chosen because it is the work most closely related to ours. The difference between this work and ours lies in the fact that the classification is static, done only one time in the beginning and the definition of interference levels thresholds are fixed and empirically defined.
- **Proportional** is similar to [14] but categorizes the interference from each profiled resource through a proportional division of the interference levels ranges (1/3 for each level, low [1%-33%], medium [34%-66%], and high [67%-100%]). This technique was chosen because this strategy to define fixed thresholds is commonly adopted in the resource management field [58,59].

One of the main challenges when classifying the interference levels of an application is to define thresholds between each level for a specific resource (for example, is 30% CPU interference low or moderate? And 60% memory interference, moderate or high?). *Even* uses an "in order" scheduling strategy, so, it does not take interference classification aspects into account and, because of that, does not need to define interference levels. *Ludwig et al.* (A) and *Proportional* (B) are similar approaches that utilize an interference classification based on fixed thresholds. In our approach, we use variable thresholds automatically defined for each resource using ML. This is one of the main contributions of this paper. To better visualize these differences, Fig. 9 depicts how the interference levels are defined for each technique. It is worth mentioning that the two dashed lines (close to the points 30 and 70, on the x-axis) in our classification approach (C), are for illustrative purposes only, and the arrows indicate that they can vary depending on the workload.

To perform the comparison, the same workloads, with the same patterns (increasing, periodic, decreasing, and constant) and over the same applications (Bench4Q, LinkBench, and TPC-H), mentioned in Section 4.2, were adopted.

As we saw in the experiments in Section 4, employing the Segmented format to classify workload interference levels can reduce interference overhead, since a classification scheme that better represents workload variations tends to use resources more efficiently. Therefore, in this experiment, we expanded this approach into dynamic classification methods that reclassify the workload after regular time intervals. More specifically, the monitored period was set to 180 s, which is a common interval found in related work that dynamically reevaluates classifications at runtime, like [29]. Classification outcomes were inserted into CIAPA over different numbers of hosts, as follows: 4, 6, 8, 10, and 12. The results are presented in Fig. 10.

In this figure, it is possible to observe that, in all executions, the *Even* method presented the worst results (higher indexes), which was already expected since this method ignores interference among applications. In general, our solution demonstrated the best placement results, presenting an improvement in the resource utilization efficiency by 27%, on average, compared to the other strategies from related work. The only exception appears with 12 hosts. In this case, each host handles only one application, producing no interference rate and generating the lowest possible interference overhead. As the number of hosts decreases, interference overhead indexes become higher. Therefore, the resource concurrency among co-hosted applications tends to increase as well. With 4 hosts, the highest indexes occurred, revealing the case with more cross-application interference incidence and greater performance degradation.

Preliminary results, with different workloads, have confirmed that resource interference is a characteristic that has a high impact on application performance, which was already demonstrated by related work. These experiments enforce that an ML-driven interference-aware dynamic classification scheme, which represents better the variability of workloads over time, can improve results even more, executing efficient scheduling decisions while enhancing the performance of applications and reducing SLA violations.

6. Related work

Many studies have been previously conducted on building interference-aware scheduling strategies, and the challenge is to have fast and scalable tools for addressing real-world applications. Furthermore, with virtualization technology, it has become possible to consolidate easily and quickly adapt resource allocation. Consequently, many recent efforts have studied performance interference issues in light of these new capabilities. In this section, we only discuss those works that are most closely related to interference-aware classification and scheduling aspects.

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. In order to reduce the effects of such issues, Ludwig et al. [14] propose placement algorithms based on interference and affinity policies and evaluate them for different workload scenarios. As a result, they achieve a reduction in response time of 10% compared to interference strategies and up to 18% when considering only affinity strategies.

Zhu and Tung [29] developed an interference model that predicts the application QoS metric. The key distinctive feature is the consideration of time-variant inter-dependency among different levels of resource interference. To prove the effectiveness of the proposed model, the authors tested several applications from a test suite and SPECWeb2005 and have achieved an average prediction error of less than 8%. In addition, it has been demonstrated that using the proposed interference model to optimize the cloud provider's metric (number of successfully executed applications) to realize better workload

placement decisions and thereby maintaining the user's application QoS.

Delimitrou and Kozyrakis [28] propose Paragon, an online and scalable data center scheduler that is heterogeneity and interference-aware. Paragon is derived from robust analytical methods and instead of profiling each application in detail, it leverages information the system already has about applications it has previously seen. It uses collaborative filtering techniques to quickly and accurately classify an unknown, incoming workload with respect to heterogeneity and interference in multiple shared resources, by identifying similarities to previously scheduled applications. Results show Paragon maintains QoS guarantees for 52% of the applications and bounds degradation to less than 10% for an additional 33% out of 8500 applications on a 1000-server cluster.

Bu et al. [30] introduce a task scheduling strategy to mitigate interference and meanwhile preserving task data locality for MapReduce applications. The authors' strategy includes an interference-aware scheduling policy, based on a task performance prediction model, and an adaptive delay scheduling algorithm for data locality improvement. The interference and locality-aware (ILA) scheduling strategy has been developed in a virtual MapReduce framework. Effectiveness and efficiency evaluation on a 72-node Xen-based virtual cluster show that ILA is able to achieve a speedup of 1.5 to 6.5 times for individual jobs and yield an improvement of up to 1.9 times in system throughput in comparison with four other MapReduce schedulers.

Zang et al. [31] propose two schedulers: one in the virtualization layer designed to minimize interference on high priority interactive services, and one in the Hadoop framework that helps batch processing jobs meet their own performance deadlines. The authors' approach uses performance models to match Hadoop tasks to the servers that will benefit them the most, and deadline-aware scheduling to effectively order incoming jobs. The combination of these schedulers allows data center administrators to safely mix resource-intensive Hadoop jobs with latency-sensitive web applications, and still achieve predictable performance for both. The evaluation shows that both schedulers allow a mixed cluster to reduce web response times by more than ten fold while meeting more Hadoop deadlines and lowering total task execution times by 6.5%.

Chen et al. [12] present CloudScope, a system for diagnosing interference for multi-tenant cloud systems. It employs a discrete-time Markov Chain model for the online prediction of performance interference of co-resident VMs. It uses the results to optimally (re)assign VMs to physical machines and to optimize the hypervisor configuration, e.g. the CPU share it can use, for different workloads. The authors have implemented CloudScope on top of the Xen hypervisor and conducted experiments using a set of CPU, disk, and network-intensive workloads and a real system (MapReduce). The interference-aware scheduler improves virtual machine performance by up to 10% compared to the default scheduler, achieving an average error of 9%. The authors claim that the hypervisor reconfiguration can improve network throughput by up to 30%.

To address latency-sensitive application issues, such as QoS impact, and overcome limitations in existing offline approaches, Shekhar et al. [18] present an online, data-driven approach that utilizes Gaussian Processes-based machine learning techniques to build predictive runtime models of the performance of the system under different levels of interference. The predictive online models are then used in dynamically adapting to the workload variability by vertically auto-scaling co-located applications such that performance interference is minimized, and QoS properties of latency-sensitive applications are met. A comparison with a representative latency-sensitive application reveals up to 39.46% lower tail latency than reactive approaches.

Wang et al. [32] developed data-driven analytical models to estimate the effect of interference among multiple Apache Spark jobs on job execution time in virtualized cloud environments. Next, they present the design of an interference-aware job scheduling algorithm leveraging the developed analytical framework. The evaluation of model accuracy was measured using real-life applications on a 6 node cluster while running up to four jobs concurrently. Experimental results show that the scheduling algorithm reduces the average execution time of individual jobs and the total execution time significantly and ranges between 47 and 26% for individual jobs and 2 to 13% for total execution time, respectively.

When addressing interference-aware scheduling, most of the above-mentioned related studies apply prediction models [12,18,28–30,32], since they perform scheduling actions based on previously measured application performance. Many of them also employ online scheduling strategies [18,28–31], since they keep profiling their job metrics at runtime to improve scheduling decisions. Our approach is different in the sense that we classify applications based on their resource usage and tell the scheduler to avoid similar stress patterns in the same machine, trying to prevent interference from happening in the first place. While [14] method uses the same general approach as we do, it uses fixed customized thresholds for the interference levels and scheduling decisions are static, valid for the entire application execution. Our approach applies variable thresholds defined for each resource without user intervention, and a dynamic ML-driven scheduler that reevaluates interference among applications during the whole execution. This can cope much better with applications that have dynamic workloads, the focus of this work.

7. Conclusion and future directions

Cloud computing has been attracting great attention from the IT community due to the promise of unlimited computing resource provisioning and the pay-per-use model. Such systems can offer these benefits through the virtualization technique, allowing large data centers to dynamically take advantage of their infrastructure while reducing energy consumption bills. Besides, cloud computing providers apply resource scheduling policies to manage their hardware efficiently, improving application performance and user satisfaction. However, multiple cloud services contending for shared resources generate cross-application interference among them, and this can lead to severe performance degradation. There are several pieces of evidence showing that interference is related to the performance penalty of application and may occur depending on the application workload and its variation.

After looking for related-studies in literature, we have evaluated how this topic has been covered recently and discovered that there are still research opportunities to explore regarding dynamic resource scheduling strategies and workload variation patterns. In this paper, we have begun to analyze dynamic resource scheduling issues based on the interference profile through the proposal of an ML-driven classification scheme. First, we have run experiments to investigate how hardware resources behave in different situations. After, we have applied an interference classification method using Unique and Segmented formats, concluding that interference levels can change over time in applications with dynamic workloads. Afterward, we have compared both classification formats and discovered that a workload-aware fine-grained classification scheme could present an improvement in scheduling decisions by 22%, on average.

Although Ludwig et al. static classification method [14] resulted in placements with better overall results than state-of-the-art, executing that method over applications with significant workload variations could produce unrepresentative classification estimates, as we showed with our Segmented format experiments. Based on these findings, in this work, we propose and evaluate a dynamic interference-aware application classifier based on machine learning techniques. Results show that our solution is able to improve scheduling decisions by 27%, on average, when compared to other approaches, thereby reducing cross-application interference at the application level in cloud computing infrastructures. Therefore, we can conclude that workload distribution is an important aspect that should be taken into account by interference-aware scheduling policies. Thus, we highlight that an interference-aware classification scheme that better represents work-

load variability can significantly reduce the interference overhead, improving resource utilization while reducing violations of Service Level Agreements.

In future work, we expect to evaluate the influence of application interference over time. We are interested in investigating the use of time-series segmentation algorithms, such as online change point detection to monitor and process data from applications as it becomes available. The goal is to detect structural changes in the data at runtime, identifying the best moment to trigger scheduling decisions, instead of using regular intervals.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgment

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brazil (CAPES) - Finance Code 001. 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. Also, this work was achieved in cooperation with HP Brasil Indústria e Comércio de Equipamentos Eletrônicos LTDA. using incentives of Brazilian Informatics Law (Law n° 8.2.48 of 1991).

References

[1] V. Priya, C.S. Kumar, R. Kannan, Resource scheduling algorithm with load balancing for cloud service provisioning, Appl. Soft Comput. 76 (2019) 416–424, http://dx.doi.org/10.1016/j.asoc.2018.12.021, URL http://www.sciencedirect.com/science/article/pii/S1568494618307105.

[2] Y. Amannejad, D. Krishnamurthy, B. Far, Detecting performance interference in cloud-based web services, in: 2015 IFIP/IEEE International Symposium on Integrated Network Management, IM, 2015, pp. 423–431, http://dx.doi.org/10.1109/INM.2015.7140319.

[3] V. Meyer, R.R. Righi, V.F. Rodrigues, C.A.D. Costa, G. Galante, C. Both, Pipel: Exploiting resource reorganization to optimize performance of pipeline-structured applications in the cloud, Int. J. Comput. Syst. Eng. (2019) http://dx.doi.org/10.1504/IJCSYSE.2019.10015444.

[4] C.T. Joseph, K. Chandrasekaran, IntMA: Dynamic Interaction-aware resource allocation for containerized microservices in cloud environments, J. Syst. Archit. 111 (2020) 101785, http://dx.doi.org/10.1016/j.sysarc.2020.101785, URL http://www.sciencedirect.com/science/article/pii/S1383762120300758.

[5] Z. Zhou, J.H. Abawajy, F. Li, Analysis of energy consumption model in cloud computing environments, in: Advances on Computational Intelligence in Energy: The Applications of Nature-Inspired Metaheuristic Algorithms in Energy, Springer International Publishing, Cham, 2019, pp. 195–215, http://dx.doi.org/10.1007/978-3-319-69889-2_10, (Ch. 1).

[6] M.G. Xavier, M.V. Neves, F.D. Rossi, T.C. Ferreto, T. Lange, 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, 2013, pp. 233–240, http://dx.doi.org/10.1109/PDP.2013.41.

[7] L.C. Jersak, T. Ferreto, Performance-aware server consolidation with adjustable interference levels, in: 31st Annual ACM Symposium on Applied Computing, SAC '16, ACM, New York, NY, USA, 2016, pp. 420–425, http://dx.doi.org/10.1145/2851613.2851625, URL http://doi.acm.org/10.1145/2851613.2851625.

[8] M.G. Xavier, Data Processing With Cross-application Interference Control via System-level Instrumentation (Ph.D. thesis), Pontifical Catholic University of Rio Grande do Sul, Porto Alegre, Brazil, 2019.

[9] Y. Zhao, R. Calheiros, G. Gange, J. Bailey, R. Sinnott, SLA-based profit optimization resource scheduling for big data analytics-as-a-service platforms in cloud computing environments, IEEE Trans. Cloud Comput. (2018) 1, http://dx.doi.org/10.1109/TCC.2018.2889956.

[10] S.K. Roy, R. Devaraj, A. Sarkar, K. Maji, S. Sinha, Contention-aware optimal scheduling of real-time precedence-constrained task graphs on heterogeneous distributed systems, J. Syst. Archit. 105 (2020) 101706, http://dx.doi.org/10.1016/j.sysarc.2019.101706, URL http://www.sciencedirect.com/science/article/pii/S1383762119305132.

[11] Y. Zhao, R.N. Calheiros, A.V. Vasilakos, J. Bailey, R.O. Sinnott, Profit maximization and time minimization admission control and resource scheduling for cloud-based big data analytics-as-a-service platforms, in: Web Services – ICWS 2019, Springer International Publishing, Cham, 2019, pp. 26–47.

[12] X. Chen, L. Rupprecht, R. Osman, P. Pietzuch, F. Franciosi, 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, 2015, pp. 164–173, http://dx.doi.org/10.1109/MASCOTS.2015.35.

[13] V. Meyer, M.G. Xavier, D.F. Kirchoff, R. da R. Righi, 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, INSTICC, SciTePress, 2019, pp. 404–411, http://dx.doi.org/10.5220/0007729004040411.

[14] U.L. Ludwig, M.G. Xavier, D.F. Kirchoff, I.B. Cezar, C.A.F. De Rose, Optimizing multi-tier application performance with interference and affinity-aware placement algorithms, Concurr. Comput.: Pract. Exper. (2019) e5098E5098, http://dx.doi.org/10.1002/cpe.5098, cpe.5098, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5098.

[15] W. Iqbal, A. Erradi, A. Mahmood, Dynamic workload patterns prediction for proactive auto-scaling of web applications, J. Netw. Comput. Appl. 124 (2018) 94–107, http://dx.doi.org/10.1016/j.jnca.2018.09.023, URL http://www.sciencedirect.com/science/article/pii/S1084804518303102.

[16] R. Nathuji, A. Kansal, A. Ghaffarkhah, Q-clouds: Managing performance interference effects for QoS-aware clouds, in: Proceedings of the 5th European Conference on Computer Systems, EuroSys '10, ACM, New York, NY, USA, 2010, pp. 237–250, http://dx.doi.org/10.1145/1755913.1755938, URL http://doi.acm.org/10.1145/1755913.1755938.

[17] Q. Zhang, L. Cheng, R. Boutaba, Cloud computing: state-of-the-art and research challenges, J. Internet Serv. Appl. 1 (1) (2010) 7–18, http://dx.doi.org/10.1007/s13174-010-0007-6.

[18] S. Shekhar, H. Abdel-Aziz, A. Bhattacharjee, A. Gokhale, X. Koutsoukos, Performance interference-aware vertical elasticity for cloud-hosted latency-sensitive applications, in: 2018 IEEE 11th International Conference on Cloud Computing, CLOUD, 2018, pp. 82–89, http://dx.doi.org/10.1109/CLOUD.2018.00018.

[19] S. Zhuravlev, S. Blagodurov, A. Fedorova, Addressing shared resource contention in multicore processors via scheduling, in: Proceedings of the Fifteenth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS XV, Association for Computing Machinery, New York, NY, USA, 2010, pp. 129–142, http://dx.doi.org/10.1145/1736020.1736036.

[20] S.K. Garg, A.N. Toosi, S.K. Gopalaiyengar, R. Buyya, SLA-based virtual machine management for heterogeneous workloads in a cloud datacenter, J. Netw. Comput. Appl. 45 (2014) 108–120, http://dx.doi.org/10.1016/j.jnca.2014.07.030, URL http://www.sciencedirect.com/science/article/pii/S1084804514001787.

[21] S. Chen, S. GalOn, C. Delimitrou, S. Manne, J.F. Martínez, Workload characterization of interactive cloud services on big and small server platforms, in: 2017 IEEE International Symposium on Workload Characterization, IISWC, 2017, pp. 125–134, http://dx.doi.org/10.1109/IISWC.2017.8167770.

[22] A.N. Toosi, C. Qu, M.D. de Assunção, R. Buyya, Renewable-aware geographical load balancing of web applications for sustainable data centers, J. Netw. Comput. Appl. 83 (2017) 155–168, http://dx.doi.org/10.1016/j.jnca.2017.01.036, URL http://www.sciencedirect.com/science/article/pii/S1084804517300590.

[23] A.M. Sampaio, J.G. Barbosa, R. Prodan, PIASA: A power and interference aware resource management strategy for heterogeneous workloads in cloud data centers, Simul. Model. Pract. Theory 57 (2015) 142–160, http://dx.doi.org/10.1016/j.simpat.2015.07.002, URL http://www.sciencedirect.com/science/article/pii/S1569190X15001069.

[24] V. Meyer, U.L. Ludwig, M.G. Xavier, D.F. Kirchoff, C.A.F. De Rose, Towards interference-aware dynamic scheduling in virtualized environments, in: D. Klusáček, W. Cirne, N. Desai (Eds.), Job Scheduling Strategies for Parallel Processing, Springer International Publishing, Cham, 2020, pp. 1–24.

[25] A. Shah, F. Wolf, S. Zhumatiy, V. Voevodin, Capturing inter-application interference on clusters, in: IEEE International Conference on Cluster Computing, CLUSTER, 2013, pp. 1–5, http://dx.doi.org/10.1109/CLUSTER.2013.6702665.

[26] L. Thamsen, I. Verbitskiy, S. Nedelkoski, V.T. Tran, V. Meyer, M.G. Xavier, O. Kao, C.A.F. De Rose, Hugo: A cluster scheduler that efficiently learns to select complementary data-parallel jobs, in: Euro-Par 2019: Parallel Processing Workshops, Springer International Publishing, 2020, pp. 519–530.

[27] R.C. Chiang, H.H. Huang, TRACON: Interference-aware scheduling for data-intensive applications in virtualized environments, in: Proceedings of 2011 International Conference for High Performance Computing, Networking, Storage and Analysis, SC '11, ACM, New York, NY, USA, 2011, pp. 47:1–47:12, http://dx.doi.org/10.1145/2063384.2063447, URL http://doi.acm.org/10.1145/2063384.2063447.

[28] C. Delimitrou, C. Kozyrakis, Paragon: QoS-aware scheduling for heterogeneous datacenters, SIGPLAN Not. 48 (4) (2013) 77–88, http://dx.doi.org/10.1145/2499368.2451125, URL http://doi.acm.org/10.1145/2499368.2451125.

[29] Q. Zhu, T. Tung, A performance interference model for managing consolidated workloads in QoS-aware clouds, in: 2012 IEEE Fifth International Conference on Cloud Computing, 2012, pp. 170–179, http://dx.doi.org/10.1109/CLOUD.2012.25.

[30] X. Bu, J. Rao, 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, HPDC '13, ACM, New York, NY, USA, 2013, pp. 227–238, http://dx.doi.org/10.1145/2493123.2462904, URL http://doi.acm.org/10.1145/2493123.2462904.

[31] W. Zhang, S. Rajasekaran, T. Wood, 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, 2014, pp. 394–403, http://dx.doi.org/10.1109/CCGrid.2014.101.

[32] K. Wang, M.M.H. Khan, N. Nguyen, S. Gokhale, Design and implementation of an analytical framework for interference aware job scheduling on apache spark platform, Cluster Comput. 22 (1) (2019) 2223–2237, http://dx.doi.org/10.1007/s10586-017-1466-3.

[33] S. Gollapudi, Practical Machine Learning, Packt Publishing Ltd., 2016.

[34] S. Athmaja, M. Hanumanthappa, V. Kavitha, A survey of machine learning algorithms for big data analytics, in: International Conference on Innovations in Information, Embedded and Communication Systems, ICIIECS, 2017, pp. 1–4, http://dx.doi.org/10.1109/ICIIECS.2017.8276028.

[35] A. Mathur, G.M. Foody, Multiclass and binary SVM classification: Implications for training and classification users, IEEE Geosci. Rem. Sens. Lett. 5 (2) (2008) 241–245, http://dx.doi.org/10.1109/LGRS.2008.915597.

[36] S. Sotiriadis, N. Bessis, R. Buyya, Self managed virtual machine scheduling in cloud systems, Inform. Sci. 433–434 (2018) 381–400, http://dx.doi.org/10.1016/j.ins.2017.07.006, URL http://www.sciencedirect.com/science/article/pii/S0020025517308277.

[37] L. Sant'ana, D. Carastan-Santos, D. Cordeiro, 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, IEEE Computer Society, Los Alamitos, CA, USA, 2019, pp. 381–390, http://dx.doi.org/10.1109/CCGRID.2019.00052, URL https://doi.ieeecomputersociety.org/10.1109/CCGRID.2019.00052.

[38] S.S. Gill, P. Garraghan, V. Stankovski, G. Casale, R.K. Thulasiram, S.K. Ghosh, K. Ramamohanarao, R. Buyya, Holistic resource management for sustainable and reliable cloud computing: An innovative solution to global challenge, J. Syst. Softw. 155 (2019) 104–129, http://dx.doi.org/10.1016/j.jss.2019.05.025, URL http://www.sciencedirect.com/science/article/pii/S0164121219301098.

[39] M. Xu, C.Q. Wu, A. Hou, Y. Wang, Intelligent scheduling for parallel jobs in big data processing systems, in: 2019 International Conference on Computing, Networking and Communications, ICNC, 2019, pp. 22–28, http://dx.doi.org/10.1109/ICCNC.2019.8685520.

[40] J. Fang, M. Wang, Z. Wei, A memory scheduling strategy for eliminating memory access interference in heterogeneous system, J. Supercomput. 76 (2020) 3129–3154, http://dx.doi.org/10.1007/s11227-019-03135-7.

[41] S.S. Manohar, H.K. Kapoor, Dynamic reconfiguration of embedded-DRAM caches employing zero data detection based refresh optimisation, J. Syst. Archit. 100 (2019) 101648, http://dx.doi.org/10.1016/j.sysarc.2019.101648, URL http://www.sciencedirect.com/science/article/pii/S1383762119304552.

[42] Y. Zhou, S. Samii, P. Eles, Z. Peng, Scheduling optimization with partitioning for mixed-criticality systems, J. Syst. Archit. 98 (2019) 191–200, http://dx.doi.org/10.1016/j.sysarc.2019.07.007, URL http://www.sciencedirect.com/science/article/pii/S1383762119301511.

[43] D. Eklov, N. Nikoleris, D. Black-Schaffer, E. Hagersten, Bandwidth bandit: Quantitative characterization of memory contention, in: Proceedings of the 2013 IEEE/ACM International Symposium on Code Generation and Optimization, CGO, 2013, pp. 1–10, http://dx.doi.org/10.1109/CGO.2013.6494987.

[44] K.J. Matteussi, C.F.R. Geyer, M.G. Xavier, C.A.F. De Rose, Understanding and minimizing disk contention effects for data-intensive processing in virtualized systems, in: 2018 International Conference on High Performance Computing Simulation, HPCS, 2018, pp. 901–908.

[45] L. Savoie, Inter-Job Optimization in High Performance Computing (Ph.D. thesis), The University of Arizona, 2019, URL http://hdl.handle.net/10150/634303.

[46] C. Iorgulescu, R. Azimi, Y. Kwon, S. Elnikety, M. Syamala, V. Narasayya, H. Herodotou, P. Tomita, A. Chen, J. Zhang, J. Wang, PerfIso: Performance isolation for commercial latency-sensitive services, in: 2018 USENIX Annual Technical Conference, USENIX ATC 18, USENIX Association, Boston, MA, 2018, pp. 519–532, URL https://www.usenix.org/conference/atc18/presentation/iorgulescu.

[47] A. Ali-Eldin, J. Tordsson, E. Elmroth, M. Kihl, Workload Classification for Efficient Auto-Scaling of Cloud Resources, Tech. Rep., Umeå University, Department of Computing Science, 2013, p. 36, URL https://webapps.cs.umu.se/uminf/reports/2013/013/part1.pdf.

[48] W. Zhang, S. Wang, W. Wang, H. Zhong, Bench4Q: A QoS-oriented E-commerce benchmark, in: IEEE 35th Annual Computer Software and Applications Conference, 2011, pp. 38–47, http://dx.doi.org/10.1109/COMPSAC.2011.14.

[49] V. Meyer, D.F. Kirchoff, M.L. da Silva, D.R. César A. F., An interference-aware application classifier based on machine learning to improve scheduling in clouds, in: 2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP, 2020, pp. 80–87, http://dx.doi.org/10.1109/PDP50117.2020.00019.

[50] D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, F. Leisch, e1071: Misc functions of the department of statistics, probability theory group, 2019, R package version 1.7-2, URL https://CRAN.R-project.org/package=e1071.

[51] R Core Team, R: A Language and Environment for Statistical Computing, R Foundation for Statistical Computing, Vienna, Austria, 2018, URL https://www.R-project.org/.

[52] S. Landset, T.M. Khoshgoftaar, A.N. Richter, T. Hasanin, A survey of open source tools for machine learning with big data in the hadoop ecosystem, J. Big Data 2 (1) (2015) 24, http://dx.doi.org/10.1186/s40537-015-0032-1.

[53] V. Meyer, D.F. Kirchoff, M.L. da Silva, C.A.F. De Rose, Interference-aware application classifier for dynamic scheduling in cloud infrastructures, J. Syst. Archit. (2021) http://dx.doi.org/10.24433/CO.3183391.v1, https://www.codeocean.com/.

[54] H. Abdi, L.J. Williams, Principal component analysis, WIREs Comput. Stat. 2 (4) (2010) 433–459, http://dx.doi.org/10.1002/wics.101, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/wics.101.

[55] C. Ferri, J. Hernández-Orallo, R. Modroiu, An experimental comparison of performance measures for classification, Pattern Recognit. Lett. 30 (1) (2009) 27–38, http://dx.doi.org/10.1016/j.patrec.2008.08.010, URL http://www.sciencedirect.com/science/article/pii/S0167865508002687.

[56] M.A. Maniar, A.R. Abhyankar, Validity index based improvisation in reproducibility of load profiling outcome, IET Smart Grid 2 (1) (2019) 131–139, http://dx.doi.org/10.1049/iet-stg.2018.0108.

[57] A. Al-Sinayyid, M. Zhu, Job scheduler for streaming applications in heterogeneous distributed processing systems, J. Supercomput. (2020) http://dx.doi.org/10.1007/s11227-020-03223-z.

[58] H. Zhou, Q. Li, W. Tong, S. Kausar, H. Zhu, P-Aware: a proportional multi-resource scheduling strategy in cloud data center, Cluster Comput. 19 (2016) 1089–1103, http://dx.doi.org/10.1007/s10586-016-0593-6.

[59] J. Krzywda, V. Meyer, M.G. Xavier, A. Ali-Eldin, P. Östberg, C.A.F. De Rose, E. Elmroth, Modeling and simulation of QoS-aware power budgeting in cloud data centers, in: 2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing, PDP, 2020, pp. 88–93, http://dx.doi.org/10.1109/PDP50117.2020.00020.

Vinícius Meyer received his bachelor's degree in Computer Engineering from the Univates University in 2014 and his master's degree in Applied Computing from the Unisinos University in 2016. Currently, he is a Ph.D. student in Computer Science at the Pontifical Catholic University of Rio Grande do Sul (PUCRS), working mainly with dynamic resource scheduling based on cross-application interference. His research interests are Distributed Systems, Cloud Computing, Machine Learning and Simulated Environments.

Dionatrã Kirchoff was born in Brazil in 1990. He received the B.E degree from the Faculdade Meridional (IMED, Passo Fundo, Brazil, 2013). He holds a specialist degree in Governance of Information Technology based on international standards from the University of Vale do Rio dos Sinos (UNISINOS, São Leopoldo, Brazil, 2015). Also, he has an M.Sc. in Computer Science from the Pontifical University Catholic of Rio Grande do Sul (PUCRS, Porto Alegre, Brazil, 2019). Since 2019 he is a Ph.D. candidate at the same university. His main areas of research interest are Resource Management, Cloud Computing, and Machine Learning.

Matheus L. Da Silva received the B.S. degree from University from Passo Fundo, Brazil, in 2017, and the master's degree in computer science from the Pontifical Catholic University of Rio Grande do Sul, Brazil, in 2020, where he is currently pursuing the Ph.D. degree with the Computer Science Graduate Program. His research interests include Parallel and Distributed Processing and Edge Computing.

César A.F. De Rose has a B.Sc. degree in Computer Science from PUCRS, a M.Sc. in Computer Science from PGCC/UFRGS and a Doctoral degree from Karlsruhe Institute of Technology (KIT - Karlsruhe, Germany). In 1998 he joined the School of Technology at PUCRS as an associate professor and member of the Resource Management and Virtualization Group (Full Professor since 2012). His research interests include several aspects of resource management, including dynamic provisioning and allocation, monitoring and profiling techniques, scheduling and optimization in parallel and distributed environments (Cluster, Grid, Cloud) and virtualization. In 2009 he founded PUCRS High Performance Computing Laboratory (LAD-PUCRS) being nowadays senior researcher.