The Journal of Systems & Software 194 (2022) 111491

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

# IADA: A dynamic interference-aware cloud scheduling architecture for latency-sensitive workloads☆

Vinícius Meyer *, Matheus L. da Silva, Dionatrã F. Kirchoff, 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

*Article history:*
Received 18 January 2022
Received in revised form 14 June 2022
Accepted 22 August 2022
Available online 26 August 2022

*Keywords:*
Interference-aware
Resource management
Dynamic workloads
Machine learning
Cloud computing

ABSTRACT

Cloud computing allows several applications to share physical resources, yielding rapid provisioning and improving hardware utilization. However, multiple applications contending for shared resources are susceptible to interference, which might lead to significant performance degradation and consequently an increase in Service Level Agreements violations. In previous work, we started to analyze resource contention and its impact on performance degradation and hardware utilization. Then, we created an interference-aware application classifier based on machine learning techniques and evaluated it comparing two classification strategies: (i) unique, when a single classification is performed over the entire applications' execution; and (ii) segmented, when the classification is carried out over multiple static-defined intervals. Moving towards a dynamic scheduling solution, we combine and improve on previous work findings and, in this work, we present IADA, a full-fledged dynamic interference-aware cloud scheduling architecture for latency-sensitive workloads. Our approach consists in improving on a segmented interference classification of applications to a dynamic classification scheme based on workload variations. Aiming at using the available resource more efficiently and respecting Quality of Services requirements, the proposed architecture was developed supported by machine learning techniques, heuristics, and a bayesian changepoint detection algorithm for online inference. We conducted a set of real and simulated experiments, utilizing a developed extension of CloudSim Toolkit to analyze and compare the proposed architecture efficiency with related studies. Results evidenced that IADA reduces by 25%, on average, the overall performance degradation.


## 1. Introduction

Due to the promise of unlimited computing resources and the pay-per-use model, many internet-based applications have started to target cloud computing infrastructures as an attracting solution (Meyer et al., 2019a). Cloud environments provide on-demand resources through the benefits of the virtualization technology for users to execute many services (Alboaneen et al., 2021). Such technology reduces operational costs in data centers by minimizing the number of hardware in use and increasing their utilization by loading more than one virtual machine (VM) instance on the same physical machine (PM). However, several cloud-applications contending for shared resources can produce cross-application interference, which may lead to considerable performance degradation and consequently to an increase in Service Level Agreement (SLA) violations (Meyer et al., 2021b). When scheduling users' services, cloud providers have to carefully place the VMs to the hosts in a way that the objectives from both providers and users would be optimized, which is a non-trivial task.

In previous work (Meyer et al., 2020), it has been presented that when there exists more than one application using the same resource (CPU, memory, network, cache, or disk), depending on the most stressed one, the resulting performance degradation index is different. For instance, by comparing applications contending for distinct hardware resources, we have observed that disk-intensive is the one that presents the highest interference levels, subsequently producing a substantial performance degradation among applications. To deeply mitigate this problem, Ludwig et al. (2019) created an interference classification based on levels for placement policies to improve resource utilization. However, their classification method was developed using fixed and empirically-defined thresholds. Aiming at tackling this issue, in previous work (Meyer et al., 2020), we introduced an interference-aware application classifier which is assisted by machine learning techniques to automatically define interference levels, and based on that, classify the applications. When compared to related studies, our classification approach demonstrates

☆ Editor: J.C. Duenas.

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

https://doi.org/10.1016/j.jss.2022.111491
0164-1212/

an improvement in placement decisions efficiency by 23%, on average, reducing resource consumption and also performance degradation at the application level.

Likewise (Ludwig et al., 2019), our classification strategy created an interference level label for each application over its entire execution. Even though the resource utilization presented overall improvements, we believed that just a single interference label for the entire application's execution cycle does not represent accurately its behavior, especially when abrupt changes may occur. Within a scenario with dynamic workloads, the hardware utilization may vary significantly, generating distinct interference rates throughout the application's execution and, consequently, directly affecting the application's degradation. Therefore, in previous work (Meyer et al., 2021b), we proposed a classification scheme that analyzes interference indexes' changes over the applications to evaluate the benefits of static-defined segmented scheduling. Preliminary results revealed an improvement in resource utilization efficiency by 27%, on average, when applying our classification approach in cloud infrastructures.

With these results, we have noticed we started moving towards dynamic interference-aware scheduling. To advance a step forward in this direction, we combine and improve on previous work findings and, in this work, we present IADA, an interference-aware scheduling architecture for dynamic workloads in cloud computing environments. The main goal is to analyze applications workloads, and based on the interference they generate, make dynamic scheduling decisions, in real-time. IADA has three principal components that profiles, analyzes, and performs scheduling decision. Aiming at using the available resource more efficiently and respecting Quality of Services (QoS) requirements, the proposed architecture was built supported by machine learning techniques, heuristics, and a bayesian changepoint detection algorithm for online inference. We compare our solution with related work using real workloads patterns and results show that IADA reduces the resulting performance degradation by 26% in real experiments, and by 24% in simulated ones. Concretely, the main contributions of this work are:

- We proposed a resource scheduling architecture observing cross-application interference aspects for dynamic workloads. Unlike previous work, which has tackled partial issues, in this study, we present a full scheduling architecture solution targeting real production systems.
- We used an online bayesian changepoint detection (OBCD) algorithm to find time-points automatically to perform classification and scheduling decisions. This specific topic was considered a gap found in previous work (Meyer et al., 2021b). Since we did not know how to find the best moments to run classification and scheduling actions, we used a static-defined interval scheme to analyze our strategy. Therefore, we included an OBCD algorithm as a new feature in the proposed architecture to overcome this issue.
- We create an optimized version of Ludwig et al. (2019) Simulated Annealing heuristic to tackle dynamic scheduling aspects. The original version, presented by Ludwig et al. (2019), was built upon static interference models, and to apply it in a dynamic scenario, we had to perform some modifications.
- We developed an extension for the CloudSim toolkit to execute interference-aware scheduling, using real case provisioning requirements and constraints, making it available in a GitHub repository[1] to allow reproducibility.
- We conducted a set of real and simulated experiments to analyze and compare the proposed architecture efficiency.

The remaining of this document is organized as follows: Section 2 discusses related work and background material. Section 3 introduces the proposed interference-aware scheduling architecture, its functionalities, and capabilities. Section 4 describes an evaluation performed to compare our solution with related studies and its results. Section 5 introduces related work in the literature. Finally, Section 6 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 present an overview of resource management and virtualization technologies. Secondly, we characterize interference and its impact on performance. Lastly, we introduce the dynamic workload application concept.

### 2.1. Resource management and virtualization

In data centers, orchestration systems need highly elastic and scalable infrastructures that allow the dynamic allocation of different resources (such as compute, storage, networking, software, or a service) in the right location and, in short, times, enable the deployment of applications (Tosatto et al., 2015). The elasticity in cloud environments is obtained by abstracting physical resources from an underlying layer through virtualization. There are different virtualization technologies, but the two most relevant in the cloud computing landscape are *Hardware virtualization* and *System-level virtualization*:

- *Hardware virtualization* (Hypervisors) abstracts the underlying hardware layers to enable complete operating systems to run inside the hypervisor as if they were an application. Paravirtualization solutions (Xen[2]) and hardware virtualization solutions (KVM[3]), in combination with hardware-specific support, integrated into modern CPU (Intel VT-x and AMD-V), can achieve a low level of overhead due to the new layer added between the virtual instance and the hardware.
- *System-level virtualization* (Containers) is based on fast and lightweight process virtualization and allows to tie up an entire application with its dependencies in a virtual container that can run on every Linux distribution. It provides its users an environment as close as possible to a standard Linux distribution. Since containers are more lightweight than VMs, the same host can achieve higher densities with containers than with VMs. This approach has radically decreased both the start-up time of instances and the processing and storage overhead, which are typical drawbacks of Hypervisor-based virtualization (Rosen, 2014).

Containerization is the state-of-art virtualization of the cloud platform (Merkel, 2014). Containers only need seconds to bootstrap, initiate, versus minutes for a regular VM (Zhang et al., 2019) (seen in Table 1). Container technologies effectively virtualize the operating system and are becoming popular in cloud computing. By encapsulating runtime contexts of software components and services, containers improve portability and efficiency for cloud application deployment (Hu et al., 2020; Pahl et al., 2019). In addition, one container can be scaled out/in within a minute, and consequently can react immediately when encountering possible unforeseen crashes. Therefore, containers are capable of tolerating fluctuating stress and reducing overhead (Scheepers, 2014), the features that autoscaling coincidentally needs.

[1] https://github.com/ViniciusMeyer/CloudSimInterference.

[2] https://xenproject.org/.

[3] https://www.linux-kvm.org/.

**Table 1**
Comparison between container and virtual machine (Zhang et al., 2019).

| Performances | Kinds of virtualization | |
|---|---|---|
| | Container | Virtual Machine |
| Size | Megabytes | Hundreds Megabytes |
| Start time | Seconds | Minutes |
| Management overhead | Low | High |
| Portability | High | Low |

Due to the characteristics presented above, we used container technology to implement the virtualization layer in this work. In the next paragraphs we present the main container solutions available in the literature, namely OpenVZ,[4] Linux-VServer,[5] LXC,[6] and Docker.[7]

OpenVZ is developed on top of kernel namespaces, allowing an isolated subset of resource to each container. It uses PID and IPC namespaces to reach isolation between processes from different contexts. OpenVZ also implements network namespaces. Moreover, it also provides different network operation modes, such as Route-based, Bridge-based and Physical-based. The main distinction between them lies at operation layer. While Route-based works in Layer 3 (network layer), Bridge-based works in Layer 2 (data link layer) and Physical-based in Layer 1 (physical layer). In the Physical-based mode, it is possible to assign a real network device (such as eth0) to a container, improving the network performance (OpenVZ, 2022).

Instead of using namespaces, Linux-VServer implements its own kernel mechanisms to provide process, network and CPU isolation. The system limits the scope of the file system from different processes through the traditional chroot system call and prohibits unwanted communications between them by using a technique called global PID space. Since it is impossible to re-instantiate processes with the same PID, Linux-VServer does not implement usual virtualization techniques, such as live migration, checkpoint and resume. Also, it does not virtualize the network layer, so that all network subsystems are shared among the containers and also with the host system (Xavier et al., 2014).

Like OpenVZ, LXC uses kernel namespaces to guarantee isolation among container instances. LXC implements PID, IPC, File System and Network namespaces. Furthermore, it also offers different types of network configurations, namely: Route-based and Bridge-based. Resource management is only performed via cgroups. With cgroups it is possible to define network configurations, limiting the CPU usage and accomplishing isolation among processes from different containers contexts. By default, LXC adopts the CFQ scheduler to control I/O operations (Menage, 2022).

Similar to LXC, Docker shares Linux cgroups, namespaces, and the Linux kernel. Although Docker was originally developed over LXC, over time Docker incorporates its own environment engine, called *libcontainer*. Different from LXC, where each container contains its own operating system, Docker provides an environment consisting of only one guest operating system, on which all processes run packed in containers, with each application having its own isolated environment (Docker Engine Overview, 2022).

A Docker application container packages a single process or application, while a LXC system container simulates a full operating system and allows users to run multiple processes simultaneously. Docker provides separate components, while LXC delivers a full solution of libraries, applications, databases, and so on. In addition, it is possible to use LXC to create different user spaces and isolate all processes belonging to each userspace, which is not what docker is intended for. In addition, LXC performs live migration without modifications. Therefore, due to all cited advantages, LXC was the container implementation used in this work.

### 2.2. Performance interference

With the resource sharing techniques evolution, each cluster node can host several applications. However, when multiple services intensively use a specific resource simultaneously, resource contention issues will occur. This problem is labeled as performance interference and may lead to severe performance degradation (Chen et al., 2015).

Virtualization technologies and server consolidation are the main drivers of high resource utilization in modern data centers (Meyer et al., 2019b). The authors of Jersak and Ferreto (2016) state that applications are affected by virtual machines that use the same resource intensively in the same physical machine and each resource is affected differently. CPU intensive applications led to performance degradation of 14%. 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.

Not only hardware virtualization is affected by performance interference, but container-based environments are as well. Disk-intensive applications running over containers promote performance degradation that uses different resources intensively. The authors of Xavier (2019) have tested a bunch of co-hosted workload combinations. While some of these combinations led to performance degradation up to 38%, there are those which cause no interference indexes. In Shah et al. (2013), the authors claim 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%.

Both works (Xavier, 2019; Shah et al., 2013) focus on analyze the interference from co-hosted applications in cluster environments. While (Xavier, 2019) presents a novel interference instrumentation tool, Shah et al. (2013) introduces an approach to correlate the performance behavior of applications running side by side. In this work, to accomplish our goal, we use both strategies (interference instrumentation and performance analysis), and on top of that, we include algorithms to efficiently schedule applications across cluster nodes.

### 2.3. Dynamic workload applications

In data centers, applications may present a variety of workload patterns, and QoS demands. Non-interactive batches are an example that requires completion time, while transactional web services are concerned with throughput guarantees. Different application workloads require a diverse type and amount of resources. For instance, batch jobs tend to be relatively stable, unlike latency-sensitive, which tends to be highly unpredictable and bursty in nature (Garg et al., 2014). Besides, latency-sensitive applications can include short latency-critical user-facing tasks, responding to web requests, for example. Also, this workloads' type can be characterized by short deadlines in the order of tens of milliseconds (Chen et al., 2017).

Multi-tenancy services need to efficiently manage resources within and among data centers taking time-varying demands into account (Iqbal et al., 2018). Their workload is not deferrable, and

4 https://openvz.org/.
5 http://www.linux-vserver.org.
6 https://linuxcontainers.org/.
7 http://www.docker.com.

this means that every time a request is received, the response should be generated immediately afterward. Consequently, such applications must perform real-time scheduling of the load, ensuring the quality of requests flow (Toosi et al., 2017). This kind of application presents an unpredictable intensity variation of resource utilization at run time due to the user's different request patterns and periodicity (Iqbal et al., 2018). Therefore, latency-sensitive applications and multi-tenant services are ideal candidates for evaluating interference effects suffered by dynamic workloads and will be considered target applications in this work.

Garg et al. (2014) creates a scheduling mechanism to guarantee the meeting of users' QoS requirements, according to SLAs specifications. They state that it is important to be aware of different types of SLAs and the mix of workloads for better resource provisioning. Results present an improvement in reducing SLA violations. Sampaio et al. (2015) address the resource allocation issues running different application workloads types (CPU-, and network-intensive ones). After performing experiments with synthetic workloads, results indicate that the authors' strategy can fulfill contracted SLAs of real-world scenarios while reducing energy expenses.

In Ebadifard and Babamir (2021), authors developed an autonomous load balancing method to alleviate the communication overhead among servers. Based on the resources, requests are divided into CPU-bound and I/O-bound. Results using dynamic workloads indicate that this proposed algorithm can distribute the workload among them equally and allocate requests to appropriate VMs based on the required resources, decreasing the communication overhead. In Daraje and Shaikh (2021), authors developed a hybrid approach combining vertical and horizontal scaling to increase resource utilization and better adapt to user requests. The results demonstrate that the proposed approach is more efficient in comparison with the existing ones.

Works (Garg et al., 2014; Sampaio et al., 2015; Ebadifard and Babamir, 2021; Daraje and Shaikh, 2021) propose scheduling strategies to improve the use of computational resources. Although these works have the same goal as we have, we apply an interference-aware scheduling, not just considering resource capacities in our solution. Besides, we developed our solution on top of a container-based cluster, while they explored traditional VMs. Furthermore, rather than just evaluating our work through simulation, as they did, we also performed experiments in a physical environment to validate the simulation and show the scalability of our solution.

### 2.4. Scheduling approaches

To evaluate the efficacy of the proposed architecture, we compared our solution to three schedulers from the literature: EVEN, CIAPA, and Segmented:

- **EVEN** implements the *EvenScheduler* algorithm, which is the Apache Storm[8] default scheduler. This algorithm distributes computation tasks across nodes in a Round-Robin manner (Al-Sinayyid and Zhu, 2020). 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. Although not interference-aware, we have decided to use this method as a baseline 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. In this case, applications are placed into the cluster nodes in a round-robin fashion, meaning that they are not moved during the experiment execution.
- **CIAPA** (Ludwig et al., 2019) evaluates the profile of the application workloads and uses a static interference classification with three levels. Its classification is static, done only one-time in the beginning of the execution using an average of an application generated interference over the entire execution. The definition of interference levels is done used fixed thresholds that are empirically defined. In a first phase, applications are placed in a round-robin manner, and after a 10-min interval, the collected data is analyzed and only one scheduling movement is done, not changing the placement of the applications after that.
- **Segmented** (Meyer et al., 2021b) applies a pseudo-dynamic interference classification with levels, similar to CIAPA. The difference is that the Segmented scheduler arbitrarily divides the applications' executions into four parts (proportional), and based on that division, classifies the generated interference of an application per segment, considering some level of change during execution. This allows the first placement to be changed three times during execution. The goal of this approach is to classify applications' interference considering the workload variability, not only using a simple average over the entire execution.

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

## 3. Dynamic interference-aware scheduling architecture

Performance interference is known to adversely impact QoS properties of applications and dynamic service demands with workload profiles further raise the challenges for cloud service providers in managing resources on-demand to satisfy SLAs while minimizing operational costs (Nathuji et al., 2010). Therefore, any solution that addresses these challenges requires an approach that should account for the workload variability and the performance interference (Shekhar et al., 2018). Due to the dynamic nature of the process, some questions come up, such as: How to classify applications in real-time based on the interference they generate? When to execute the classification? When to schedule them and how to tradeoff migration costs?

Finding a solution that comprehensively covers all the mentioned issues is not a straightforward task. Recently proposed approaches present significant improvements regarding interference classification and dynamic scheduling strategies. However, there are still gaps in the state-of-the-art. One example, is the lack of a complete scheduling architecture that automatically handles dynamic workloads, finding the best intervals to classify and schedule applications among cluster nodes. Besides, this architecture should address performance interference aspects without earlier workload know-how and with no user mediation.

Standing for the concept that dynamic interference-based scheduling algorithms, which analyzes workload variations over time, could improve even more resource utilization, and consequently reduce SLA violations, in this section, we propose IADA, a full-fledged dynamic interference-aware cloud scheduling architecture for latency-sensitive workloads. This architecture aims to efficiently schedule applications based on the interference they generate, without user intervention and with no previous workload knowledge. The main goal is to analyze hardware events and supported by classification, time interval detection, and scheduling algorithms, to find the best applications' placement set at runtime.

In the next subsections, we introduce the proposed architecture, describing its capabilities and functionalities in detail.

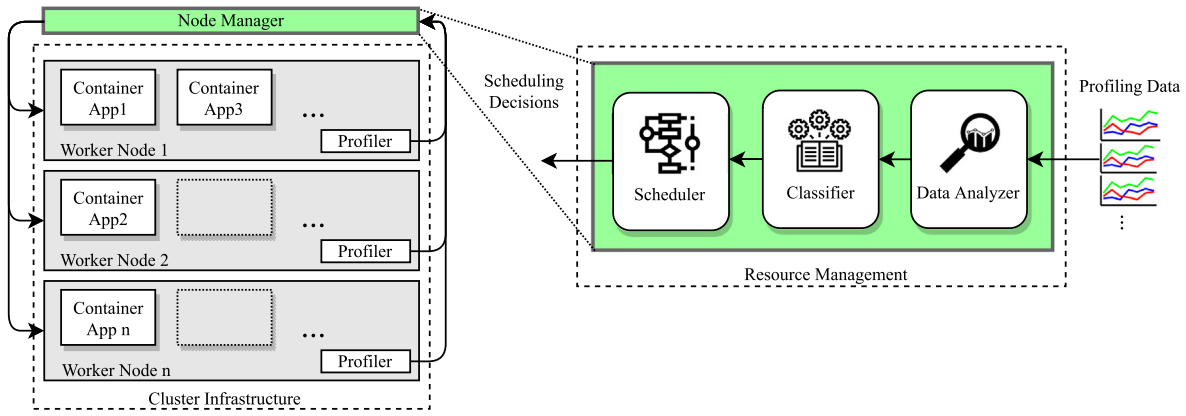


**Fig. 1.** System architecture.

### 3.1. Proposed architecture

Usually, interference-aware task schedulers are performed by combining three main steps (Chiang and Huang, 2011; Zhu and Tung, 2012; Bu et al., 2013; Zhang et al., 2014; Xavier, 2019; Wang et al., 2019): (i) profiling queued tasks based on their resource needs; (ii) predicting the performance interference; and (iii) scheduling the task on the best-suited node, which is the node that causes the lowest performance interference effects. Since we are interested in scheduling real-time applications based on the workload variability, we decide to use a reactive approach. For this reason, we adjusted the prediction step by splitting it into two more ones: (ii-A) classifying interference and (ii-B) analyzing the best time intervals from applications at runtime in order to perform the scheduling (next) step.

Therefore, to build a dynamic interference-aware scheduling architecture, we used four main components, presented as follows: (i) a profiler that reads hardware metrics; (ii) a technique that gets significant workload changes, based on profiling data at runtime; (iii) an interference classification method supported by a combination of machine learning techniques; and (iv) a scheduling algorithm that interprets all data generated by the previous components and makes efficient placement decisions.

The choice of using a reactive technique is normally adopted before applying a proactive one, however as a matter of fact we intend to investigate and compare proactive approaches as well in the future. To perform the proposed architecture, all these aforementioned components were assigned in a node that works over the entire computational environment analyzing and executing scheduling decisions, referred here as Node Manager. Also, an interference profiler module is executed inside each cluster node, profiling all applications and sending all data to the Node Manager. First, these metrics are received and analyzed by the Data Analyzer component, which is responsible for examining and finding abrupt changes in the application's behavior. After, these metrics are also sent to the Classifier component that has the duty of classifying each application in a given period, defined by the previous component, into interference levels. Then, the Scheduler module performs hardware orchestration decisions by running an algorithm based on generated data from the two previous components. Fig. 1 presents an overview of the proposed architecture, distinguishing each layer.

The Node Manager is continually monitoring and analyzing potentially interference information from the cluster infrastructure. It is worth noting that the Profiler module is always monitoring the entire infrastructure while feeding the Data Analyzer and Classifier components. While both mentioned modules analyze and classify the received data, when they find there is room to make scheduling decisions, they send that information to the Scheduler module to apply it over the cluster infrastructures. Fig. 2 depicts the architecture data flow, where it is possible to observe how collected metrics are processed through each component.

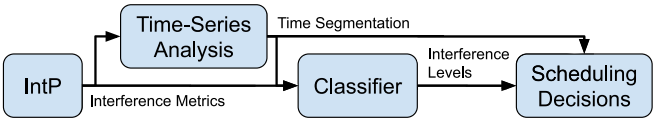


**Fig. 2.** Architecture data flow.

This cycle will always run while there exists more than one application running in the cluster. To clarify, let us take an example: suppose that application1 starts on Node1, together with application2. In a given moment, application1 and application2 become to contend for CPU (or another resource, i.e.), and the Node Manager perceives it and decides to migrate the container which runs application2 to Node2. This scheduling action aims to use the resources more efficiently, improving QoS and consequently reducing SLA violations. In the next subsections, each component is presented in detail.

#### 3.1.1. Interference profiler

Profiling runtime applications is not a straightforward task, given that different tasks may burst arbitrary resources, causing variation in resource consumption. In addition, an intrusive profiler can induce the performance of applications and compromise the reliability thereof. The literature presents works that care about resource contention aspects among applications in a simple way, existing or not (Ludwig et al., 2019). Also, a number of application profiling mechanisms, ranging from kernel-based (Linux Trace Toolkit Project Page, 2002) to runtime (Urgaonkar et al., 2003) profiling that uses especially linked libraries, have been proposed in the past. Recently, a tool called IntP (Xavier, 2019) has been developed, an open-source system-level monitoring tool which analyzes selected architectural counters and operating systems data structures to estimate the stress an application puts on each hardware's subsystem and consequently infer the potential interference it could generate in other applications hosted in the same physical machine. Different from tools that apply a more high-level approach using micro benchmarks and application metrics, IntPs low level instrumentation enables a more accurate prediction of the performance degradation that results from contention on shared resources, with less monitoring overhead.

IntP is subdivided into modules responsible for each access method on specific resources at the infrastructure level and outcomes the percentage of their utilization relative to the total

system capacity, for each running application. This isolated measurement provides analytical information to infer how much an application could potentially interfere with other applications consolidated in the same physical machine, so that conflicts can be avoided by the scheduler. More specifically, the tool provides 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;

IntP is used in this work to profile applications at runtime (every second), so it is possible to perform an analysis on how interference is potentially hurting consolidated applications over time, and trigger a new scheduling operation if needed.

### 3.1.2. Time-series analysis

We aim to evaluate the influence of application interference over time, but dealing with dynamic workloads at runtime is a challenging task. Whether we wish to perform scheduling decisions in an online trend, time is an important factor that must now be considered in our model. For example, to perform dynamic scheduling actions on the fly, it is necessary to define at what moments our architecture will execute them. As already mentioned, in previous work (Meyer et al., 2021b), we moved a step forward and created a static-defined time interval scheme to start analyzing segmented scheduling, and preliminary results presented a considerable improvement in hardware efficiency. Since we are interested in accomplishing automatic scheduling decisions based on interference levels generated across applications, we need to carry out a statistical time-series analysis that deals with the profiled data and points trend patterns out. However, some questions come into play when working with time series, such as: Is this data stationary? Is there seasonality? To work around these questions, we performed an online change point analysis (Pagotto, 2019) aiming at determining the time points with the most significant behavior changes, considered crucial for analyzing and classifying the profiled data, and subsequently, performing scheduling actions.

Change points are abrupt variations in the generative parameters of a data sequence. Online detection of change points is useful in modeling and prediction of time series in application areas such as finance, biometrics, and robotics (Pagotto, 2019). A time series consists of multiple assessments of a specific outcome measure, at group level, at regularly spaced time intervals. The "interruption" or "change point" of the time series is an identifiable real-world event.

Since IntP profiles each application in an isolated manner and outcomes multiple metrics (different resources) from each one, we first had to reduce its dimensionality. To do that, we apply the Principal Component Analysis (PCA) (R Core Team, 2019) over each application. PCA is a dimensionality-reduction method that is often used to reduce the dimensionality of datasets, by transforming a set of variables into a smaller one that still contains most of the valuable information in the large dataset. In our case, we decided to reduce seven metrics profiled from IntP to only one for each application. Depending on the order the algorithm sort those metrics, the PCA outcome changes. So, to find out what is the best order to arrange interference metrics, we have performed several tests and decided to place those metrics in an order that follows its performance degradation priority. In previous work (Meyer et al., 2021b), we introduced such priority order,

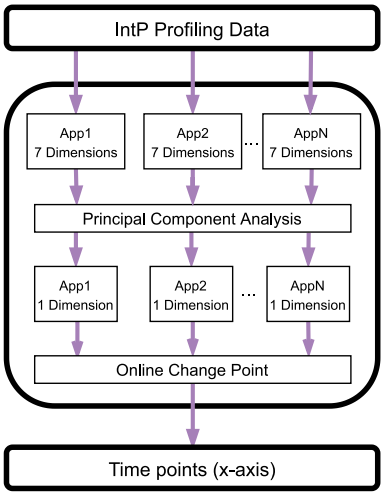


**Fig. 3.** Data scheme of data profiling (IntP), dimensionality reduction (PCA), and discovering change points over time (OCPD).

presenting that when there is resource contention incidence, some hardware components present more elevated performance degradation indexes than others, so that we decided to apply PCA with the following resource order: disk, memory, cpu, cache, and network. This means that performance degradation caused by disk resource contention is bigger than caused by network, for example. If there will be no disk usage, PCA takes the next resource in the queue order, memory in this case. If there will be no memory utilization, the next resource will be considered as the principal, and so on.

After reducing the profiled data from each application to a single dimension, observing its performance degradation priority, we apply the Online Change Point Detection (OCPD) function, from R Package (Pagotto, 2019), overall applications' metrics. Such a technique provides an implementation of Bayesian online change point detection that handles multivariate data, computing the set of change points with the highest probability in an online way (updating the results with each incoming point). This method outputs a list of change points over time (x-axis) during running the model, in an online fashion. The entire process of reducing and analyzing profiled data is depicted in Fig. 3.

To present a simple use case example, we run an experiment adopting Node-Tiers.[9] 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. First, we choose two memory-intensive applications from the Node-Tiers suite, then we created a synthetic workload for each one. On purpose, each workload produced an interval with a high-load request rate: (A) between 60 and 120 s; and (B) between 180 e 240 s, accordingly Table 2.

Both applications (A and B) were executed together while profiled with IntP, and the results are presented at the top of Fig. 4.

The collected metrics passed through the PCA phase and then produced A and B resulting data, depicted at the bottom of the same Figure. Finally, this data was submitted to the OCPD function, returning the moments where both applications presented abrupt behavior changes (60 s, 120 s, 180 s, and 240 s), seen in the

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

**Table 2**
A and B applications' workloads behavior.

| Intervals (s) | A (req/s) | B (req/s) |
|---|---|---|
| 0–60 | 100 | 100 |
| 61–120 | 200 | 100 |
| 121–180 | 100 | 100 |
| 181–240 | 100 | 200 |
| 241–300 | 100 | 100 |

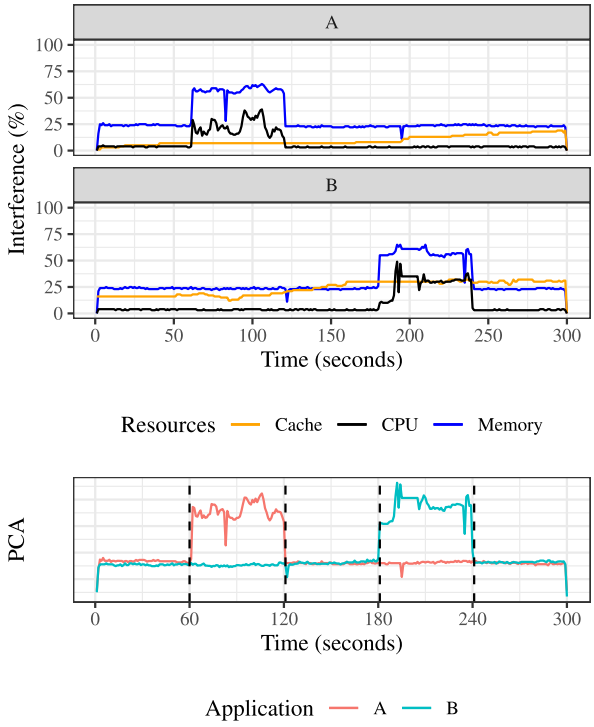


**Fig. 4.** Profiled data (IntP) from A and B execution (top); PCA resulting data along with found OCPD change points (bottom).

same image. It is possible to observe that OCPD handles multiple applications due to its *multivariate* characteristics, being a good candidate function for our architecture.

### 3.1.3. Interference classification

A number of techniques have been proposed regarding interference classification, such as: collaborative filtering (Delimitrou and Kozyrakis, 2013), decision-tree (Moreno et al., 2013; Javadi and Gandhi, 2017), major interference source (Kumar and Setia, 2017; Devarajan et al., 2018) and resources historic mean (Caglar et al., 2014; Caglar et al., 2016). The authors from Ludwig et al. (2019), the most closely related to our study, developed a scheduling model that considers interference levels among applications to increase resource usage. Even though the authors' approach increases the state-of-the-art in the scheduling resource field, the proposed classification was developed with fixed thresholds, empirically defined.

Aiming at finding out alternatives to minimize interference overhead effects over scheduling decisions, in previous work (Meyer et al., 2020; Meyer et al., 2021b), we have proposed a classifier that quantifies cross-application interference in levels over time, standing for the concept that an interference classifier method that better represents the workload variability improves hardware utilization. The main purpose of our classification method is to return the hardware resources' interference produced by applications, within a time slice, to a given degree. This is achieved by exploiting the combination of two different machine learning algorithms: (i) SVM for classification and (ii) K-Means for clustering. Initially, SVM receives interference data from applications, 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. More specifically, we adopted four interference levels: (i) *absent*, when there is no interference incidence; (ii) *low*; (ii) *moderate*; and (iv) *high*. Both machine learning algorithms use a training dataset, previously defined, to assist their decisions. More details about the classification method are presented in Meyer et al. (2021a).

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. This classification process is repeated until the end of the execution, characterizing the dynamicity of our approach, where interference levels are reevaluated regularly, accordingly to OCPD function (seen in Section 3.1.2), so that we are able to better react to significant changes in the workload.

To present an example, we use a decision support benchmark called TPC-H.[10] This benchmark evaluates the performance of various decision support systems by the execution of sets of queries against a standard database under controlled conditions. Also, we create an increased workload, starting with a low load and gradually going to a high load. This workload execution was profiled with IntP, arbitrarily divided into four segments, and each one was classified by our approach. The classification result is depicted in Fig. 5.

It is possible to notice that there are resources that do not change their labels, for instance, memory, disk, 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 Cache cases.

The CPU has a smooth increase in its behavior, moving from moderate to high label. In addition, Cache keeps its labels with the highest levels while executing, changing from high to moderate, and from moderate to high interference levels again. This highlights that, due to the dynamic workload nature, the application execution present different interference labels during its execution.

### 3.1.4. Scheduling algorithm

Scheduling consists of ordering running jobs across available computational resources (Hu et al., 2020; Thamsen et al., 2020). To do this with interference awareness, first, the Node Manager pulls the tasks into available node slots. After, it profiles the interference from each application and, based on the information generated on previous components, suits them on the best candidate nodes that minimize the overall performance interference. IADA is an architecture that relies mainly on a reactive approach so that the applications are constantly profiled and when the OCPD technique finds significant workload variations, the most recent interval data is used to perform scheduling decisions.

The applications start at time zero and they are monitored continuously every second. The time-series analysis evaluates the data and returns $P$, which is the point found by the OCPD function with the greatest representativeness in the workload variation of the applications (according to Section 3.1.2). When $P$ is found, the classification module generates an interference label for each application resource running in each container. The

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

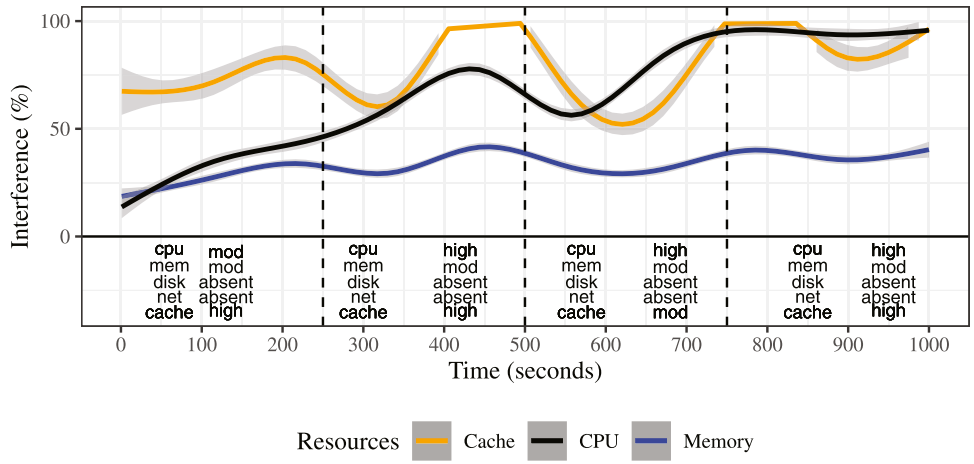


**Fig. 5.** Segmented TPC-H static interference classification. To facilitate the visualization, a Loess function was applied to smooth short-term variations in each resource. Resources labels that changed are shown in bold in the bottom plot. IntP metrics that do not suffer any interference were not depicted.

interval between $P_{(n-1)}$ and $P_n$ is defined as $\Delta T_n$. When a $\Delta T_n$ is found, the scheduling is performed based on the most recent data, which means, the last $\Delta T_n$ outcome.

The traditional view for real-time scheduling problems focuses on how to find a feasible schedule for an application set. However, the scheduling of a given application set is not a straightforward task. With the rapid increase in the use of powerful cloud systems, an efficient task scheduling policy, which deals with the assignment of tasks to resources, is required to reduce performance degradation. Task scheduling is an established NP-Hard optimization problem that can be effectively tackled with metaheuristic algorithms (Chhabra et al., 2020). Taking this statement into account, we decided to use a heuristic algorithm to solve our problem. Ludwig et al. (2019) tested many heuristics to schedule applications with interference awareness and concluded that Simulated Annealing (SA) presented the best overall results. So, in this work, we decided to apply a modified SA algorithm that addresses interference-aware aspects.

The SA algorithm is an optimization method that mimics the slow cooling of metals, which is characterized by a progressive reduction in the atomic movements that reduce the density of lattice defects until a lowest-energy state is reached (Kirkpatrick et al., 1983). Similarly, the simulated annealing algorithm generates a new potential solution to the problem by altering the current state, according to a predefined criterion. The new state solution is then based on the satisfaction criterion and may be accepted even if they do not lead to an improvement in the objective function.

Since our architecture moves applications among cluster nodes at runtime, we developed an algorithm based on SA to find the best applications' arrangement set in order to minimize performance degradation. The algorithm 1 presents how our architecture scheduling policy works.

Initially, the algorithm creates an application set $S$, in which each container receives one application instance to execute. All containers are distributed among cluster nodes by a *RoundRobin* function that receives a set of physical machines $P$ and a set of applications $A$ to be executed. Every SA iteration generates one new solution $S_{modified}$ that is compared to the best solution at that point. This new solution is generated by the Random Swap Function, presented in Algorithm 2.

This function relies on a randomized approach, in which the function has a 50% chance of swapping random applications in the cluster and a 50% chance of swapping the application of the cluster node with the highest score to the cluster node with the lowest score.

**Algorithm 1:** Optimized Simulated Annealing

```
Data: P, A, temperature, coolingRate
Result: solution_best
s = roundRobin(P,A);
bestsolution = s;
while (temperature > 1) do
    newsolution = randomFunction(s);
    bestscore = bestsolution.getInterferenceScore();
    newscore = newsolution.getInterferenceScore();
    if (newscore < bestscore ) then
        if (bestsolution.getMig() < newsolution.getMig() )
         then
            bestsolution = newsolution;
        end
    end
    temperature *= 1 - coolingRate;
end
```

**Algorithm 2:** Random Swap Function

```
Data: solution
Result: S_modified
p = Math.random();
if (p < 0.5) then
    app_1 = getRandomApp(solution);
    app_2 = getRandomApp(solution) ;
else
    app_1 = getHigherScoreApp(solution);
    app_2 = getLowerScoreApp(solution);
end
swap(app_1, app_2, solution);
```

After finding the new solution, if $S_{modified}$ presents an interference degradation index lower than the current one, the algorithm replaces the current (best) solution with the new one. To compare both solutions, we create a function *InterferenceScore()* that analyzes the interference levels in $\Delta T_n$ and returns a total interference score, which is calculated by using the function seen in Eq. (1).

$$TotalIntScore_{\Delta T} = \sum_{k=1}^{N} IntScore_{Host}, \quad \forall k \in s \quad | \quad k \geq 1. \tag{1}$$

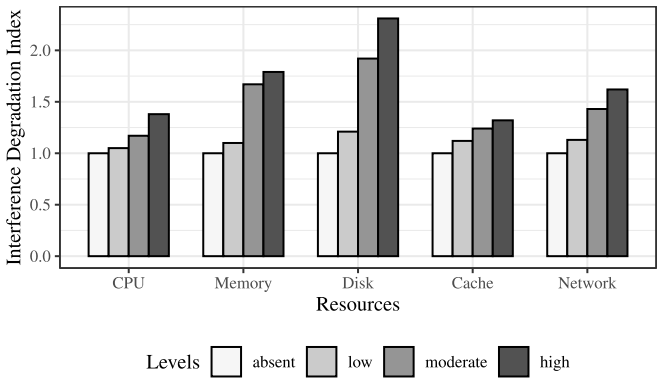


**Fig. 6.** Interference degradation index by resource.

The total interference score is the result of the sum of all interference scores from each cluster node, where $k$ represents the hosts' number in the environment. Each cluster node has its own interference score as well, this score is calculated with a function demonstrated by Eq. (2).

$$IntScore_{Host} = \begin{cases} \prod_{j=1}^{N} IntScore_{App}, & \text{if } j \geq 2 \\ 0, & \text{otherwise} \end{cases} \quad (2)$$

where, $j$ denotes the applications' number running in each cluster node, ranging from 2 to $N$ (total number of applications). If there exists less than 2 applications running in a cluster node, it will not generate interference incidence in that specific node and consequently will return a zero-score, since only one or no one application does not cause interference. Finally, the application interference score is calculated by Eq. (3).

$$IntScore_{App} = cpu(L) \times mem(L) \times disk(L) \times net(L) \times cache(L) \quad (3)$$

All resource interference metrics (*cpu*, *memory*, *disk*, *network*, and *cache*) were measured and allocated into an level $L$. Depending on the level they are set, the interference overhead index value varies, according to Fig. 6.

To find these Interference Degradation Indexes (IDI), first, we ran applications with each resource-intensive (e.g. CPU-intensive, memory-intensive, and so on) in isolation and took the average response time. After, we ran each one again co-hosted with one more application instance at a specific level (low, moderate, and high), according to the classifier method, and found the average response time from both. With those metrics we discovered how much each resource degraded at each interference level by using Eq. (4).

$$IDI = \frac{ResponseTime_{(level+absent)}}{ResponseTime_{absent}} \quad (4)$$

To illustrate this scenario for memory, let us take an example: We executed a memory intensive application in isolation, which resulted in $ResponseTime_{absent} = 23.2$ ms. While co-hosting with a Low-intensive memory application, the runtime increased to $ResponseTime_{(low+absent)} = 24.4$ ms, which gives IDI of 1.10. When co-hosting with a Moderate-intensive application, the response time increases to $ResponseTime_{(moderate+absent)} = 39.2$ ms, resulting in an IDI of 1.69. Finally, when co-hosting with a High-intensive memory application, the response time increased to $ResponseTime_{(high+absent)} = 41.5$ ms, resulting in an IDI of 1.79.

Another important aspect that is analyzed in the SA algorithm, is the number of migrations done with the newly generated solution. If the number of migrations performed in the new solution is bigger than the best solution, this new solution is disregarded and another one is considered. The migrations number is taken with the help of the *getMig*() function, as seen in algorithm 1.

## 4. Evaluation and results

In this section, we describe how the experiments were conducted, the scope, and the limits of the project. Also, the details about workload, application, and the computational environment adopted in this work are discussed.

### 4.1. Application and workload

To investigate applications that present dynamic workload (unpredictable load variation) by stressing different hardware resources, Node-Tiers (seen in Section 3.1.2) has been adopted. This tool explores the latency-sensitive application's concept (client–server) and allows the creation of workload variations. The goal is to stress hardware resources in many ways (distinct resources) through many latency-sensitive applications from this suite benchmark, increasing and decreasing the request arrival rate, executing scheduling decisions at runtime, handling changes in the workload. Node-Tiers tool also provides an intensive-data pressure to the target server/cluster. Its disk and memory applications stress hardware resources likewise real intensive-data workloads do.

To create a most realistic scenario, we evaluated our architecture using three real-world workload traces. The first one is from the Wikimedia project, found in Wikipedia[11] traces. Specifically, we collected the page view statistics for the main page in the English language for the month of January 2021. The second one is from Alibaba Open Cluster Trace,[12] this one is sampled from one of Alibaba production clusters. There are both online services and batch workloads, and we collected only information from Sigma, the online service scheduler. The last one is from NASA[13] dataset, consisting of all web requests made to the 1998 World Cup Web site between April 30, 1998, and July 26, 1998. Although this particular workload is not considered as a newer one, it remains being adopted by recent studies (Mallikharjuna Rao and Rama Satish, 2022; Radhika and Sudha Sadasivam, 2021; Chhetri et al., 2021).

In addition to these workloads being widely used in related work in the resource management field, they were chosen also because they are not drawn from synthetic functions or independent distributions, but rather represent real-world traces that exhibit the realistic patterns of a workload resulting from user-, program- and operating system behaviors. In addition, they are lengthy traces obtained over extended periods of time, allowing us to evaluate if the proposed architecture can cope with dynamic real time applications, while improving resource utilization.

### 4.2. Experiments scenarios

To explore the efficiency of our architecture, we dived all experiments into two phases: first, we use a real-scenario with a small number of machines to ensure all proposed steps work correctly together and to guarantee the simulation phase outcomes are in accordance to the reality, reflecting reliable results; and second, based on the previous phase, we build a simulated environment, to test our architecture with a bigger number of cluster nodes, and consequently, more applications. In the next sections we describe how each phase was performed and its results.

[11] https://dumps.wikimedia.org/other/analytics/.
[12] https://github.com/alibaba/clusterdata/tree/master/cluster-trace-v2018.
[13] ftp://ita.ee.lbl.gov/html/contrib/.

#### 4.2.1. Real experiments

To run our experiments within a real testbed, we used a cluster called Pantanal that belongs to the LAD Laboratory[14] from PUCRS. This cluster has Dell PowerEdge R740xd nodes, each one equipped with: 2x Intel Xeon Gold 5118 Processor, 300 GB DDR4 RAM Memory, 1TB Hard Drive, and 4x Gigabit Ethernet Interface. Also, as the Node Manager, we used one Dell Optiplex 990 outside of the cluster, equipped with 8 GB of RAM, and one Core i5 processor.

To stress different resources subsystems (CPU, memory, disk, network, and cache), we used different applications from the Node-Tiers suite. The server-side was performed over the cluster, while the client-side was configured on a single computer, using Artillery[15], a load stress testing tool. The server-side applications were executed inside containers, more specifically one container per application. Since containers present many benefits concerning traditional virtual machines (seen in Section 2.1), we decide to adopt LXC/LXD containers as target virtualization technology, allowing us to schedule the applications with live migration facilities across the cluster nodes with Checkpoint/Restore In Userspace (CRIU[16]) functionalities. All pieces of equipment were connected through a Gigabit Ethernet Network. We ran four applications inside each cluster node, and each one was submitted to a period of 2-h workload trace (randomly chosen), mixing the elected datasets and creating greater variation among application workloads.

To evaluate the proposed architecture, we compared our work to the references presented in the related work section that apply similar scheduling strategies and target the same type of environments and applications so that a direct comparison to our results is possible, namely CIAPA (Ludwig et al., 2019) and Segmented (Meyer et al., 2021b). We also included EVEN – not cited in related work section because it does not consider interference aspects – that uses a round-robin scheduling strategy, as a baseline. This strategy is widely applied nowadays by schedulers such as Apache Storm, for example.

For this experiment phase, we used four nodes of the Pantanal cluster, each one executing four Node-tiers applications, totaling 16 applications. When using latency-sensitive applications, the response time (latency) metric quantifies how long the user must wait for a response to a query, regardless of the quality of the response. Together with data quality metrics, latency metrics provide the best indication of the end-user experience under normal conditions and during outages (Broadwell, 2004). For this reason, we decided to use the *Average Response Time* as the main performance metric in this work, which represents the total latency for the test divided by the number of requests submitted to the server-side by the users-side. The response time was collected from each application during the entire experiment with Artillery, and their total average is presented in Fig. 7.

It is worth noting that in all experiments, our proposed architecture presented the best results, improving in average the response time by 26% when compared to CIAPA, EVEN, and Segmented scheduling approaches. Also, it is interesting to note that EVEN scheduler reaches the higher response time indexes (worst results), as predicted since this scheduling strategy is not interference-optimized.

By running 16 applications with a mix of workloads variations, IADA detected 12-time points with an expressive (global) behavior change. Consequently, 12 periods were classified and each one provoked scheduling actions. As mentioned before, the Data Analyzer component uses a bayesian change point detection to find workload behavior modification, but this does not imply that all applications, in all analyzed intervals, had their interference labels modified, exchanging their interference degree (levels). The applications only have their labels modified if the workload variation has an abrupt alteration.

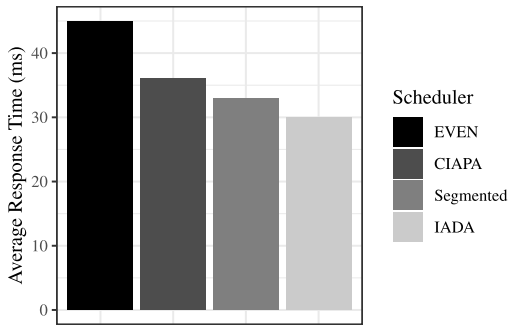


**Fig. 7.** Average Response Time from real experiments phase.

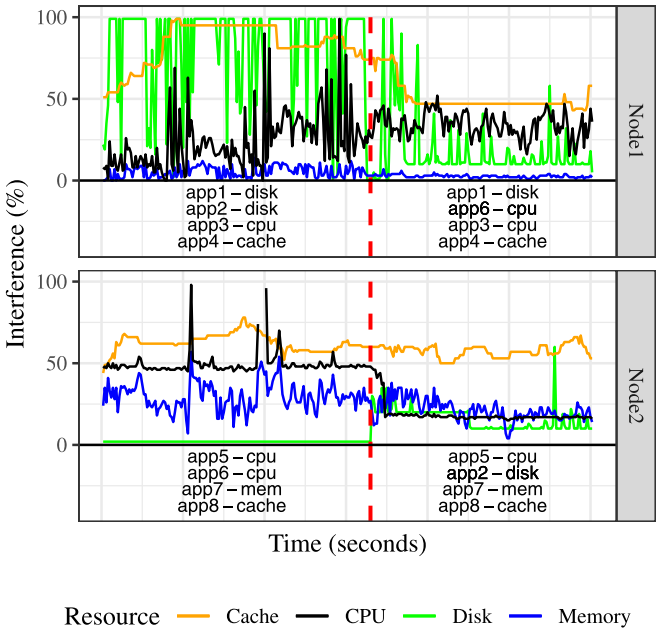


**Fig. 8.** Average interference indexes in Node1 and Node2 while performing a scheduling action. Applications which have migrated across nodes are shown in bold. *This image presents only data from 2 of 4 nodes used in the real experiment.

To give an example of how much the interference in a node is affected by scheduling actions, we collected the average interference generated in Node1 and Node2 within a given period, every second, while a scheduling rearrangement was performed, and presented in Fig. 8.

This image illustrates the average of interference generated by the 8 applications running in Node1 and Node2 before and after a scheduling movement. The red dashed line demonstrates the exact moment the scheduling was executed.

By looking at the interference measures, it is possible to perceive two interesting facts: (i) after the scheduling, Node1 and Node2 exchanged app2 (disk intensive) and app6 (cpu intensive) applications (in bold at the image); and (ii) after this rearrangement, Node1 had its disk interference ratios considerably reduced while Node2 had its disk ratios increased. Also, Node2 had its cpu interference indexes reduced while Node1 had its overall cpu usage increased. In general, it is possible to observe that this

14 https://www.pucrs.br/ideia/lablad/.
15 https://artillery.io/.
16 https://criu.org/.

scheduling operation provided a balance across interference indexes, improving the resources' usage and reducing applications' response time.

Therefore, these results show that a technique that analyzes frequently the generated interference over time is able to reduce the overall system's overhead, using the infrastructure more efficiently, and consequently, improving QoS requirements. Also, this experiments show that the proposed architecture presents interesting and trustworthy outcomes, and they will be used to calibrate and perform the simulation experiments, presented in the next section.

### 4.3. Simulated experiments

In order to carry out experiments closer to a real scenario, a large physical machine set is necessary, and to have more flexibility to perform different host arrangements, we decide to scale our approach out through simulation as well. First, we search in the literature for tools that simulate cloud infrastructures (Lim et al., 2009; Kliazovich et al., 2012; nez et al., 2012; Calheiros et al., 2011). After exploring each simulation tool, we conclude no one of them offers an environment that handles interference aspects from applications. Then, we have confirmed that CloudSim (Calheiros et al., 2011) is the most widely spread cloud simulator and by far also the most sophisticated. It is developed as an add-on-top of the grid network simulator GridSim (Casanova, 2001). CloudSim is a completely customizable tool that supports modeling, creation of one or more VMs, and mapping tasks to appropriate virtual machines. This gives CloudSim the ability to handle a complex simulation environment. It mainly targets application developers or testers as it gives the ability to configure several variables such as the number of users, data centers, and cloud resources along with the location of both users and data centers. Besides many other studies extending CloudSim, such as Beloglazov and Buyya (2012), Guérout et al. (2013), Xavier et al. (2017) and Krzywda et al. (2020), there is one in particular that supports Container as a Service (CaaS), namely ContainerCloudSim (Piraghaj et al., 2017). This extension provides a platform for modeling and simulating containerized cloud computing environments. Therefore, it is the most fitting simulation tool to use nowadays and the one we have chosen to extend with applications interference features. We have developed the CloudSimInterference plug-in, a trace-driven extension to the CloudSim simulation tool. First, each application was monitored in a physical machine (previously), and all IntP metrics were kept and used as input to our workload simulations. This means that we did not use the workload trace itself, but rather the resulting resource utilization levels from a real scenario. Since we measured all interference levels from all resources (seen in Section 3.1.4), we used those metrics to perform scheduling actions instead of generating them within the simulation tool.

Because bandwidth sharing is not considered by default in CloudSim, we introduced a migration degradation overhead to overcome this limitation. This overhead was measured in isolated physical machines, considering several migration scenarios, with many different workloads. Each time the simulator performs a migration, this overhead is added to the total interference cost (*TotalIntScore*), being considered in the simulated experiments.

To implement our simulation plugin, we have made many classes modifications in ContainerCloudSim. First, we extend the *containerCloudlet()* class to *InterferenceContainerCloudlet()*. This class is responsible for representing the application behavior, and we include all Interference metrics measured with IntP inside them, each one keeps the information from each application trace generated from real execution. Another major modification, was the integration with the R algorithms to perform the OCPD and ML functions, presented in the Sections 3.1.2 and 3.1.3. To perform this integration, we include the JRI (Java-R-Integration) library on the java side and the rJava library on the R side. Also, we extended *containerDatacenter()* class to *InterferenceContainerDatacenter()*, including several functions to handle the modifications done with interference metrics utilization.

To generate a considerable number of applications (*InterferenceContainerCloudlets*) for the simulation experiments, we have executed several hours of each workload trace (seen in Section 4.1) with five applications instances from Node-Tiers suite, stressing the main hardware resources (cpu, memory, disk, net, and cache). After, we randomly divided those execution traces collected with IntP into two-hour segments to use as input data in our simulation experiments.

To run the simulated experiments, we use four different arrangements of cluster node numbers and application instances, presented in Table 3.

**Table 3**
Hosts/application arrangements used in simulated experiments.

| Hosts | Applications |
|---|---|
| 6 | 24 |
| 12 | 48 |
| 24 | 96 |
| 48 | 192 |

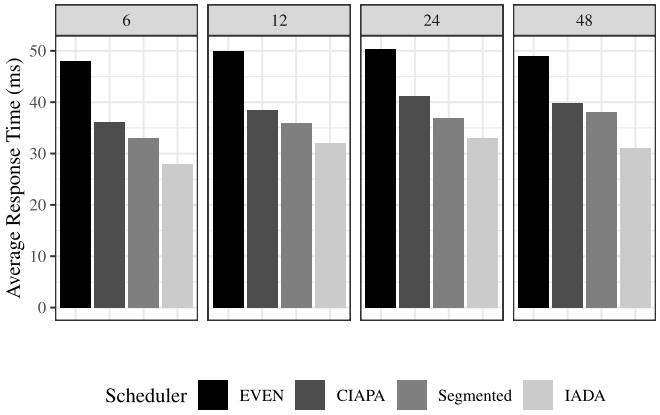


**Fig. 9.** Average Response Time from simulated experiments phase in each host arrangement (6, 12, 24, and 48).

As mentioned before, we used real experiments to calibrate our simulator. To produce more reliable results, compatible with real case scenarios, we applied the same number of applications in each cluster node, as done in practical experiments phase (four application instances per node). All results shown in this section display the average over 10 simulation trials with 95% confidence interval level. Fig. 9 presents the results from the simulated experiment's phase.

It is noteworthy that, in all experiments, IADA achieves the best results (lower indexes). When compared to EVEN scheduler, our proposed architecture reached a reduction of 37% on average in the response time. Not surprisingly, EVEN reaches the worst results as well, similar to the real experiments, because this scheduler was not developed based on interference-awareness. Compared to the CIAPA approach, IADA reduced the average response time in 21% and, in contrast to Segmented, the state-of-the-art strategy, IADA obtained a reduction of 14% in the average response time.

To analyze how much the response time varies over time, we took the average response time in each scheduled interval, during the experiments running with 24 nodes (96 applications),

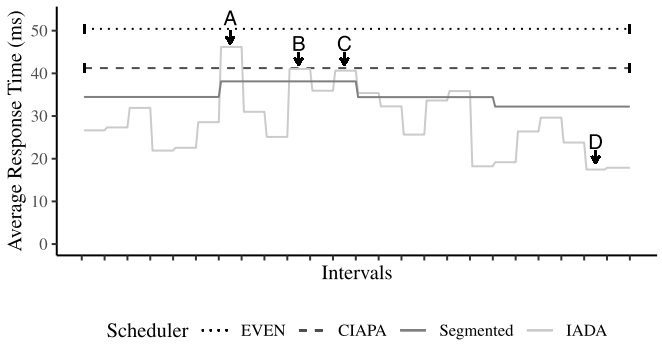


**Fig. 10.** Average Response Time in each scheduled interval from the 24-nodes experiment.

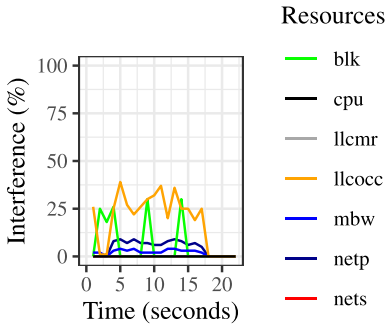


**Fig. 11.** Resources' behavior while running a container migration across cluster nodes.

and compare them with EVEN, CIAPA, and Segmented schedulers' results. These results are presented in Fig. 10.

It is interesting to observe that EVEN scheduler places the applications at the beginning of the execution and after that, they are not rearranged anymore. That is the reason its representation in the image is a straight line, depicting the total average in the entire execution. Something similar happens with the CIAPA strategy, within the first 300 s the interference metrics are collected and analyzed, and after that just one placement decision is taken, not being executed again, and its representation also is a straight line, representing the total average in the entire execution. In the Segmented approach, the application execution was divided into four segments, and at the end of each one, scheduling actions were taken. This strategy improves the overall response time when compared to EVEN and CIAPA strategies because it adjusts the infrastructure to applications, taking into account the variability of workloads (Meyer et al., 2021b), even considering only a few segments (four in this case).

In general, IADA reaches the lowest response time rates (best results). However, there was an interval that presented worst results than CIAPA's scheduler, for example, depicted in point A. This happened because IADA relies on a reactive approach, using the most recent data and not a general view to make scheduling decisions, so that in interval A the workload had an abrupt behavior change, not presenting the best scheduling arrangement, but still is considered as an acceptable result.

There were intervals that IADA touches CIAPA's outcomes, which were the B and C intervals' cases. Also, the major response time reduction can be seen in the interval D, reaching an average improvement of 57% in relation to EVEN, CIAPA, and Segmented scheduling approaches.

It is important to highlight that the proposed scheduling architecture adjusts applications over the hardware based on the workload oscillation, in real-time, and in a reactive manner. So far, the outcomes found in this work support our idea that an interference-aware dynamic scheduling architecture designed to observe workloads tends to reduce the overhead generated by cross-application interference over the system, and consequently utilizing the available hardware resources more efficiently.

### 4.4. Overhead evaluation

Considering the dynamic nature of the problem, it was necessary to run some preliminary steps in order to make the scheduling decisions. As mentioned in Section 3, those steps are profiling applications, analyzing time-series data, and performing interference classification with ML techniques. After that, with a heuristic-oriented algorithm, it was possible to execute scheduling actions.

**Table 4**
Number of intervals found by Data Analyzer component and how many migrations were performed per host arrangement.

| Hosts | App. | Intervals | Migrations | Mig./Interval |
|---|---|---|---|---|
| 6 | 24 | 18 | 115 | 6 |
| 12 | 48 | 23 | 245 | 11 |
| 24 | 96 | 24 | 712 | 30 |
| 48 | 192 | 27 | 1323 | 49 |

All these techniques put some overhead pressure on the cluster system, as well as on the Node Manager. To examine these aspects, in the next sections, we performed some analysis to find out if the overhead generated by the proposed architecture could make its use infeasible.

#### 4.4.1. Migration

When running experiments within the real scenario, we performed many container migrations across the cluster. In terms of hardware resource usage, the overhead rate created by a single LXC/LXD migration can be considered low over the entire computational environment. To present how much this operation affects the system, we executed one container migration and profiled LXC/LXD processes with IntP. Fig. 11 illustrates the hardware utilization while performing a single container migration across the cluster.

In our experiments the mean migration time was about 18 s, depending on the resource the applications is stressing more, it can take more or less time to conclude this operation.

However, when the number of container migrations increases, the resulting overhead can considerably increase as well. So, such operations should be minimized as much as possible. As mentioned in Section 3.1.4, IADA uses a heuristic to find the best applications' set to schedule applications over the cluster. We developed an optimized version of CIAPA (Ludwig et al., 2019) scheduler algorithm, taking the number of migrations into account when deciding the best scheduling actions. IADA scheduling algorithm was developed to dynamically deals with workload behavior changes, adjusting its decisions considering the most recent data and its behavior. So that it is important to keep the number of migration operations at the minimum, and for this reason, the amount of migrations operations is contemplated as a quality measure to decide if the new solution created is better than the actual one.

Considering IADA automatically finds the best moments to classify data intervals, and based on that, it runs scheduling decisions, in all experiments we also observe the number of intervals found by the Data Analyzer component and how many migration actions were performed. These metrics are presented in Table 4 for each host arrangement.

Observing this table, it is evident that the more applications IADA is controlling, the more intervals will be found. The reason this happens is that the more data (more applications with distinct workload patterns) the proposed architecture is analyzing, the greater the amount of information to be processed, consequently increasing the dynamism in the environment and generating greater optimization opportunity. Of course, this is also highly dependent on the variation of workloads, but since we are monitoring dynamic applications, these are not unexpected outcomes. Looking at the number of migrations performed, it is noticeable that the more applications running in the cluster, the greater the number of migrations as well, practically following a linear trend with applications' number. At the first look, the number of migrations performed with 192 applications seems to be exaggerated, but when dividing the number of migrations by the interval (Mig./interval), it is possible to notice that the number of migrations makes sense, being proportional in relation to the number of hosts, almost one migration per host per interval, demonstrating a reasonable outcome.

#### 4.4.2. Machine learning

According to Section 3.1.4, when $P$ is defined, then $\Delta_T$ is established. After, this data interval is sent to the Classifier component so that depending on the data quantity (interval length), this process time can vary. On average, in our experiments, each application took about 1 s to be classified. To improve the performance of this proceeding, we used the *doParallel* library (Corporation and Weston, 2020) from R packages. This library can be adopted to send tasks (encoded as function calls) to each of the processing cores on a machine in parallel. This is done by using a function that distributes the tasks to multiple processors. After, this function gathers the responses up from each process call and it returns a list of responses, which is the same length as the list or vector of input data (one return per input item). To speed the classification process up, we performed the ML training phase previously, generating *.RDA* files. These files are the results from the R programming language within the training dataset phase so that it is not necessary to execute the model training phase each time the classification process is performed.

Depending on the number of applications running inside the cluster and the length of data sent to the Classifier component, the ML analysis can take longer to be performed. In our experiments, the largest classified interval took less than 25 s, which is very reasonable, since each scheduling decision was not performed within less than a 20-s interval, that is the meantime to migrate containers across the cluster nodes.

To measure the overhead of the classification process even with a high number of running applications, we performed a scalability experiment with this component. Therefore, we created a set of application workloads with varied interval lengths between 30 and 600 s, testing short, medium, and long periods. For each interval, we gradually increased the number of running application workloads (24, 48, 96, and 192) to ensure that the execution time of the classification (y axis) is not significant even for many applications (colors) executing in a long interval of time (x axis), what would invalidate its use in a dynamic environment. Fig. 12 presents these results.

When looking at this figure, it is possible to observe the classification follows a linear trend, which is already expected since the classification is not a distributed process, and at certain times we are allocating more tasks than the number of cores our Node Manager owns. It is interesting to notice that it takes less than 30 s to accomplish the classification of 192 application workloads with a 10-min interval length, meaning the biggest application quantity with the largest period in this experiment. This result can be considered acceptable if the target applications do not have workloads with extreme behavior patterns variability.

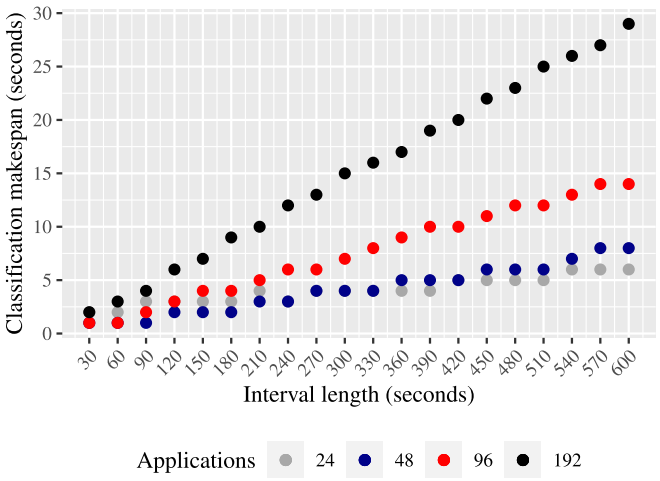


**Fig. 12.** Classification makespan from a set of applications with different interval lengths. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

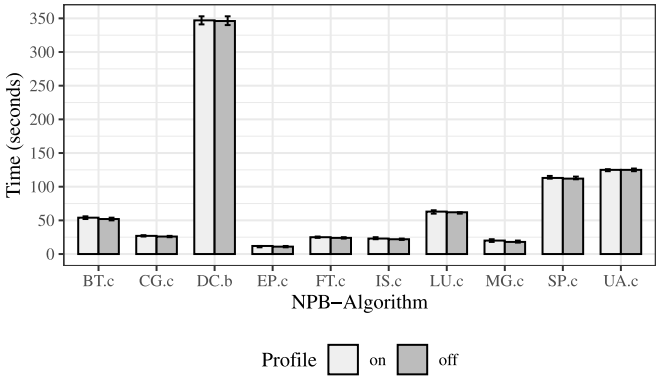


**Fig. 13.** NAS Parallel Benchmark (NPB) algorithms' executions with and without IntP profiling them.

Summing it up, if there will be performed an expressive number of applications or the length of the monitored period will be increased, it could be necessary to adopt a different machine with more computational power (more CPUs) than ours as Node Manager, in order to ensure the architecture works correctly.

#### 4.4.3. Profiler

To enforce IntP does not input a considerable overhead over the hardware, we have run an experiment with NAS Parallel Benchmarks (NPB),[17] which are a small set of programs designed to help evaluate the performance of parallel supercomputers. First, we ran BT.d, CG.c, DC.b, EP.d, LU.c, MG.d, and UA.c algorithms without any profiler. After, we ran each one again with IntP profiling them. Each execution was performed 10 times and the resulting meantime of them are presented in Fig. 13.

Since IntP core works with low-level kernel events instrumentation, in our experiments, when it was enabled, its execution practically did not generate overhead. Meaning that IntP plays a non-intrusive role over the entire system.

## 5. Related work

Virtualization technology enables highly scalable services to be easily delivered by cloud providers within different contexts.

[17] https://www.nas.nasa.gov/publications/npb.html.

However, many real applications present dynamic workloads and need to be managed to avoid interference problems, minimizing performance degradation in resource-shared environments. Recently, several research efforts were conducted addressing interference-aware scheduling strategies, and in this section, we present them and provide a contrast to our work.

Bu et al. (2013) introduce a task scheduling strategy to mitigate interference and meanwhile preserve 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. Results show that the authors' proposal 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. Zhang et al. (2014) 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 evaluation shows that both schedulers allow a mixed cluster to reduce web response times by more than tenfold while meeting more Hadoop deadlines and lowering total task execution times by 6.5%.

Chen et al. (2015) present CloudScope, a system that diagnoses interference for multi-tenant cloud sources. It employs a discrete-time Markov Chain model for the online prediction of performance interference of co-resident VMs. 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 also claim that the hypervisor reconfiguration can improve network throughput by up to 30%. Melo Alves et al. (2018) have developed an interference-aware virtual machine placement strategy for HPC applications in cloud computing. The authors' approach implements a method that predicts interference levels in order to minimize the number of used physical machines. Results presented that the authors' method reduced interference by more than 40%, using the same hardware set.

Shekhar et al. (2018) present an online, data-driven approach to build runtime predictive performance models. 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. (2019) developed data-driven analytical models to estimate the effect of interference among multiple Apache Spark jobs on job execution time in virtualized cloud environments. Experimental results show that the scheduling algorithm reduces the average execution time of individual jobs (between 47 and 26%) and the total execution time (2 to 13%).

Ludwig et al. (2019) propose placement algorithms based on interference levels for different workload scenarios. As a result, they achieve a 10% reduction in response time compared to interference strategies. Evolving the author's efforts, in previous work (Meyer et al., 2021b), we propose a machine learning-driven classification scheme for dynamic interference-aware resource scheduling in cloud computing environments. We have presented how a classification approach, that better represents the workload variations, affects resource scheduling. By analyzing how hardware resources react to different applications, we explored distinct interference classification formats and evaluate their efficiency, taking the dynamic nature of cloud workloads into account. Then, we applied an interference-aware application classifier (Meyer et al., 2020) based on machine learning techniques and compare it with related work, adopting a variety of workload patterns. Preliminary results revealed an improvement of 27% hardware utilization efficiency when applying our classification approach in cloud data centers.

Our work differs from related studies due to evolution of technology and the update of operating systems and Kernels versions. Such evolution makes possible to extract information from hardware in a manner that was not possible in the recent past. One example of that, is the advanced feature developed by Intel, called Intel Cache Monitoring Technology (CMT), now it is entirely viable to collect information about the usage of the cache by applications running inside any piece of equipment. This is something essential since multi-thread architectures are in an exponential growth within the computer market. This technology allows us to use an ID denominated Resource Monitoring ID (RMID) to metrify the number of threads scheduled among the operation system. For each thread, there is one ID associated with it, therefore, those metrics can be collected within an MSR interface. Something that could not be possible before the creation of this technology.[18]

Bu et al. (2013) study is limited to analyze only CPU from hypervisor through *xentop* counters and disk metrics through linux *iostat* from hadoop workloads. Similar to Bu et al. (2013) work, Zhang et al. (2014) proposes ILA, in which only CPU and disk counters are monitored from hadoop applications. Likewise our work, Chen et al. (2015) includes in Cloudscope strategy some different components to distribute systems' responsibilities. Also, authors' approach profiles specific virtual machines characteristics, such as VCPUs and VNICs. The main difference from ours architecture is that we run applications over containers instead of traditional virtual machines, as mentioned before, this kind of virtualization presents many benefits over the traditional method, such as low management overhead and portability (Zhang et al., 2019). Melo Alves et al. (2018) work utilizes a slowdown factor that measures the applications' time and how much (in percent) each one increased its time regarding isolated execution. Also, an average period is calculates over each host to compare them with each other. Our work is different in the sense that we apply interference levels instead of the raw percent of performance degradation, and on top of that we also use automatic techniques that outcome the interference levels, with no user intervention.

Similar to our work, Shekhar et al. (2018) investigate the interference generated by container-based instances with workload variations. However, our work is distinctive in the following ways: (i) instead of focusing on a single server, our proposed approach is performed over a distributed architecture (cluster). Further, our proposed architecture focuses exclusively on applications that have dynamic workloads, such as latency-sensitive applications, while authors mix them with batch-job ones. Wang et al. (2019) are interested in improving Apache Spark jobs' makespan. Since this type of application is workflow-oriented, they present multiple jobs' stages, and the authors analyze each stage separately, stating that each one has different (specific) resources behavior. The authors consider only execution time, CPU usage, disk I/O rate, and network I/O rate, not observing cache and memory metrics, which our approach does.

## 6. Conclusion and future directions

Cloud service providers offer many services through virtualization techniques to users over the Internet. As many virtual machines (VMs) run on the same computational node, they share physical resources, and consequently there exists great opportunity to produce resource contention, which results in applications' performance degradation. Therefore, how to place VMs

[18] https://github.com/intel/intel-cmt-cat.

to reduce performance degradation and guarantee QoS requirements is still a challenging task. Recently, many related studies have proposed strategies to tackle these issues, but none of them consider cross-application interference aspects to dynamically make scheduling decisions.

In previous work (Meyer et al., 2020; Meyer et al., 2021b) we proposed a machine learning-driven classification scheme for dynamic interference-aware resource scheduling in cloud computing environments. It has been presented how a classification approach, that deals better with workload variability, affects resource scheduling. Preliminary results revealed an improvement in resource utilization efficiency by 27%, on average, when applying this classification approach in cloud scenarios.

In this work, we further analyze this topic and develop IADA, a full-fledged interference-aware scheduling architecture for dynamic workloads in clouds. IADA combines and improves different techniques studied in previous work, including machine learning, bayesian algorithms, and heuristics to find abrupt changes in applications' workload behavior, classifying and placing them in a way that minimizes the overall resource contention. We compare our solution with close-related studies in this field using real workloads from NASA, Wikimedia, and Alibaba Open Cluster Trace datasets and results show that IADA reduces the resulting performance degradation by 26% when compared to EVEN, CIAPA, and Segmented scheduling approaches in real experiments, and by 24% in simulated experiments.

Moreover, we also performed and presented an overhead analysis under the Migration, Machine Learning, and Profiler techniques used by IADA and we have concluded that: the scheduling algorithm was developed and optimized to reduce as much as possible the number of migrations. Our solution presents reasonable numbers of scheduling actions per interval, keeping the general overhead at a acceptable rate; the machine learning techniques used generate a layer of overhead, but in our experiments, these indexes are considered acceptable. However depending on the number of nodes and applications the architecture is going to control, the resulting overhead could be bigger than ours, and consequently, the Node Manager might be resized; the chosen profiler (IntP) practically does not put any overhead pressure over the system, since this tool was built to analyze hardware events at the kernel layer.

In future work, we expect to evaluate proactive scheduling approaches by applying machine learning prediction algorithms and comparing them with the current work. The goal is to analyze how much the performance degradation can be reduced by forecasting the workload variability and anticipating the hardware resource's arrangement within a dynamic scheduling architecture.

## CRediT authorship contribution statement

**Vinícius Meyer:** Conceptualization, Methodology, Software, Investigation, Data curation, Writing. **Matheus L. da Silva:** Review, Validation, Writing. **Dionatrã F. Kirchoff:** Review, Validation, Writing. **Cesar A.F. De Rose:** Conceptualization, Methodology, Validation, Supervision.

## 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.

## Acknowledgments

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

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

Alboaneen, D., Tianfield, H., Zhang, Y., Pranggono, B., 2021. A metaheuristic method for joint task scheduling and virtual machine placement in cloud data centers. Future Gener. Comput. Syst. 115, 201–212. http://dx.doi.org/10.1016/j.future.2020.08.036.

Beloglazov, A., Buyya, R., 2012. Optimal online deterministic algorithms and adaptive heuristics for energy and performance efficient dynamic consolidation of virtual machines in cloud data centers. Concurr. Comput.: Pract. Exper. 24 (13), 1397–1420. http://dx.doi.org/10.1002/cpe.1867.

Broadwell, P.M., 2004. Response time as a performability metric for online services. In: Report No. UCB//CSD-04-1324. Computer Science Division (EECS),University of California, Berkeley, California 94720, pp. 1–49.

Bu, X., Rao, J., Xu, C.-z., 2013. 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, pp. 227–238. http://dx.doi.org/10.1145/2493123.2462904.

Caglar, F., Shekhar, S., Gokhale, A.S., 2014. Towards a performance interference-aware virtual machine placement strategy for supporting soft real-time applications in the cloud. In: REACTION 2014, 3rd IEEE International Workshop on Real-Time and Distributed Computing in Emerging Applications, Proceedings, Rome, Italy. December 2nd, 2014. pp. 15–20.

Caglar, F., Shekhar, S., Gokhale, A., Koutsoukos, X., 2016. Intelligent, performance interference-aware resource management for IoT cloud backends. In: 2016 IEEE First International Conference on Internet-of-Things Design and Implementation (IoTDI). pp. 95–105. http://dx.doi.org/10.1109/IoTDI.2015.36.

Calheiros, R.N., Ranjan, R., Beloglazov, A., De Rose, C.A.F., Buyya, R., 2011. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. Softw. - Pract. Exp. 41 (1), 23–50. http://dx.doi.org/10.1002/spe.995.

Casanova, H., 2001. Simgrid: a toolkit for the simulation of application scheduling. In: 1st IEEE/ACM International Symposium on Cluster Computing and the Grid. pp. 430–437.

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

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

Chhabra, A., Singh, G., Kahlon, K.S., 2020. Multi-criteria HPC task scheduling on iaas cloud infrastructures using meta-heuristics. Cluster Comput..

Chhetri, M.B., Forkan, A.R.M., Vo, Q.B., Nepal, S., Kowalczyk, R., 2021. Exploiting heterogeneity for opportunistic resource scaling in cloud-hosted applications. IEEE Trans. Serv. Comput. 14 (6), 1739–1750. http://dx.doi.org/10.1109/TSC.2019.2908647.

Chiang, R.C., Huang, H.H., 2011. 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, pp. 47:1–47:12. http://dx.doi.org/10.1145/2063384.2063447.

Corporation, M., Weston, S., 2020. doparallel: Foreach parallel adaptor for the 'parallel' package. R package version 1.0.16. URL https://CRAN.R-project.org/package=doParallel.

Daraje, M., Shaikh, J., 2021. Hybrid resource scaling for dynamic workload in cloud computing. In: 2021 IEEE International Conference on Mobile Networks and Wireless Communications (ICMNWC). pp. 1–6. http://dx.doi.org/10.1109/ICMNWC52512.2021.9688556.

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

Devarajan, H., Kougkas, A., Challa, P., Sun, X., 2018. Vidya: Performing code-block I/O characterization for data access optimization. In: 2018 IEEE 25th International Conference on High Performance Computing (HiPC). pp. 255–264. http://dx.doi.org/10.1109/HiPC.2018.00036.

Docker Engine Overview, 2022. URL https://docs.docker.com/engine/.

Ebadifard, F., Babamir, S.M., 2021. Utonomic task scheduling algorithm for dynamic workloads through a load balancing technique for the cloud-computing environment. Cluster Comput. 24 (2), 1075–1101. http://dx.doi.org/10.1007/s10586-020-03177-0.

Garg, S.K., Toosi, A.N., Gopalaiyengar, S.K., Buyya, R., 2014. SLA-based virtual machine management for heterogeneous workloads in a cloud datacenter. J. Netw. Comput. Appl. 45, 108–120. http://dx.doi.org/10.1016/j.jnca.2014.07.030.

Guérout, T., Monteil, T., Costa, G.D., Calheiros, R.N., Buyya, R., Alexandru, M., 2013. Energy-aware simulation with DVFS. Simul. Model. Pract. Theory 39, 76–91.

Hu, Y., Zhou, H., de Laat, C., Zhao, Z., 2020. Concurrent container scheduling on heterogeneous clusters with multi-resource constraints. Future Gener. Comput. Syst. 102, 562–573. http://dx.doi.org/10.1016/j.future.2019.08.025.

Iqbal, W., Erradi, A., Mahmood, A., 2018. Dynamic workload patterns prediction for proactive auto-scaling of web applications. J. Netw. Comput. Appl. 124, 94–107. http://dx.doi.org/10.1016/j.jnca.2018.09.023.

Javadi, S.A., Gandhi, A., 2017. DIAL: Reducing tail latencies for cloud applications via dynamic interference-aware load balancing. In: 2017 IEEE International Conference on Autonomic Computing (ICAC). pp. 135–144. http://dx.doi.org/10.1109/ICAC.2017.17.

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

Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., 1983. Optimization by simulated annealing. Science 220 (4598), 671–680. http://dx.doi.org/10.1126/science.220.4598.671.

Kliazovich, D., Bouvry, P., Khan, S.U., 2012. GreenCloud: a packet-level simulator of energy-aware cloud computing data centers. J. Supercomput. 62, 1263–1283. http://dx.doi.org/10.1007/s11227-010-0504-1.

Krzywda, J., Meyer, V., Xavier, M.G., Ali-Eldin, A., Östberg, P., De Rose, C.A.F., Elmroth, E., 2020. 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). pp. 88–93. http://dx.doi.org/10.1109/PDP50117.2020.00020.

Kumar, R., Setia, S., 2017. Interface aware scheduling of tasks on cloud. In: 2017 4th International Conference on Signal Processing, Computing and Control (ISPCC). pp. 654–658. http://dx.doi.org/10.1109/ISPCC.2017.8269758.

Lim, S., Sharma, B., Nam, G., Kim, E.K., Das, C.R., 2009. MDCSim: A multi-tier data center simulation, platform. In: 2009 IEEE International Conference on Cluster Computing and Workshops. pp. 1–9. http://dx.doi.org/10.1109/CLUSTR.2009.5289159.

Linux Trace Toolkit Project Page, 2002. URL https://www.opersys.com/LTT/.

Ludwig, U.L., Xavier, M.G., Kirchoff, D.F., Cezar, I.B., De Rose, C.A.F., 2019. Optimizing multi-tier application performance with interference and affinity-aware placement algorithms. Concurr. Comput.: Pract. Exper. e5098. http://dx.doi.org/10.1002/cpe.5098, e5098 cpe.5098.

Mallikharjuna Rao, K., Rama Satish, A., 2022. A comprehensive study on workloads in cloud computing. In: Bianchini, M., Piuri, V., Das, S., Shaw, R.N. (Eds.), Advanced Computing and Intelligent Technologies. Springer Singapore, Singapore, pp. 505–514.

Melo Alves, M., Teylo, L., Frota, Y., Drummond, L.M.A., 2018. An interference-aware virtual machine placement strategy for high performance computing applications in clouds. In: 2018 Symposium on High Performance Computing Systems (WSCAD). pp. 94–100. http://dx.doi.org/10.1109/WSCAD.2018.00024.

Menage, P., 2022. Control groups definition, implementation details, examples and api. URL https://www.kernel.org/doc/Documentation/cgroup-v1/cgroups.txt.

Merkel, D., 2014. Docker: Lightweight linux containers for consistent development and deployment. Linux J. 2014 (239).

Meyer, V., Kirchoff, D.F., da Silva, M.L., César, D.R.A.F., 2020. 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). pp. 80–87. http://dx.doi.org/10.1109/PDP50117.2020.00019.

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

Meyer, V., Kirchoff, D.F., Da Silva, M.L., De Rose, C.A., 2021b. ML-driven classification scheme for dynamic interference-aware resource scheduling in cloud infrastructures. J. Syst. Archit. 116, 102064. http://dx.doi.org/10.1016/j.sysarc.2021.102064.

Meyer, V., Ludwig, U.L., Xavier, M.G., Kirchoff, D.F., De Rose, C.A.F., 2020. Towards interference-aware dynamic scheduling in virtualized environments. In: Job Scheduling Strategies for Parallel Processing. Springer International Publishing, Cham, pp. 1–24.

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

Meyer, V., Xavier, M., Kirchoff, D., Righi, R., De Rose, C.F., 2019b. Performance and cost analysis between elasticity strategies over pipeline-structured applications. In: Proceedings of the 9th International Conference on Cloud Computing and Services Science - CLOSER. SciTePress, INSTICC, pp. 404–411. http://dx.doi.org/10.5220/0007729004040411.

Moreno, I.S., Yang, R., Xu, J., Wo, T., 2013. Improved energy-efficiency in cloud datacenters with interference-aware virtual machine placement. In: 2013 IEEE Eleventh International Symposium on Autonomous Decentralized Systems (ISADS). pp. 1–8. http://dx.doi.org/10.1109/ISADS.2013.6513411.

Nathuji, R., Kansal, A., Ghaffarkhah, A., 2010. 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, pp. 237–250. http://dx.doi.org/10.1145/1755913.1755938.

nez, A.N., L., V.-P.J., Caminero, A.C., Castañé, G.G., Carretero, J., Llorente, I.M., 2012. iCancloud: A flexible and scalable cloud infrastructure simulator. J. Supercomput. 10, 185–209. http://dx.doi.org/10.1007/s10723-012-9208-5.

OpenVZ, 2022. URL https://openvz.org/.

Pagotto, A., 2019. ocp: Bayesian online changepoint detection. R package version 0.1.1. URL https://CRAN.R-project.org/package=ocp.

Pahl, C., Brogi, A., Soldani, J., Jamshidi, P., 2019. Cloud container technologies: A state-of-the-art review. IEEE Trans. Cloud Comput. 7 (3), 677–692. http://dx.doi.org/10.1109/TCC.2017.2702586.

Piraghaj, S.F., Dastjerdi, A.V., Calheiros, R.N., Buyya, R., 2017. ContainerCloudSim: An environment for modeling and simulation of containers in cloud data centers. Softw. - Pract. Exp. 47 (4), 505–521. http://dx.doi.org/10.1002/spe.2422.

R Core Team, 2019. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria, URL https://www.R-project.org/.

Radhika, E., Sudha Sadasivam, G., 2021. A review on prediction based autoscaling techniques for heterogeneous applications in cloud environment. Mater. Today Proc. 45, 2793–2800. http://dx.doi.org/10.1016/j.matpr.2020.11.789, International Conference on Advances in Materials Research - 2019.

Rosen, R., 2014. Linux containers and the future cloud. URL https://www.linuxjournal.com/content/linux-containers-and-future-cloud.

Sampaio, A.M., Barbosa, J.G., Prodan, R., 2015. PIASA: A power and interference aware resource management strategy for heterogeneous workloads in cloud data centers. Simul. Model. Pract. Theory 57, 142–160. http://dx.doi.org/10.1016/j.simpat.2015.07.002.

Scheepers, M.J., 2014. Virtualization and containerization of application infrastructure : A comparison. In: 21st Twente Student Conference on IT. pp. 1–7.

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

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

Thamsen, L., Verbitskiy, I., Nedelkoski, S., Tran, V.T., Meyer, V., Xavier, M.G., Kao, O., De Rose, C.A.F., 2020. Hugo: A cluster scheduler that efficiently learns to select complementary data-parallel jobs. In: Schwardmann, U., Boehme, C., B. Heras, D., Cardellini, V., Jeannot, E., Salis, A., Schifanella, C., Manumachu, R.R., Schwamborn, D., Ricci, L., Sangyoon, O., Gruber, T., Antonelli, L., Scott, S.L. (Eds.), Euro-Par 2019: Parallel Processing Workshops. Springer International Publishing, Cham, pp. 519–530.

Toosi, A.N., Qu, C., de Assunção, M.D., Buyya, R., 2017. Renewable-aware geographical load balancing of web applications for sustainable data centers. J. Netw. Comput. Appl. 83, 155–168. http://dx.doi.org/10.1016/j.jnca.2017.01.036.

Tosatto, A., Ruiu, P., Attanasio, A., 2015. Container-based orchestration in cloud: State of the art and challenges. In: 2015 Ninth International Conference on Complex, Intelligent, and Software Intensive Systems. pp. 70–75. http://dx.doi.org/10.1109/CISIS.2015.35.

Urgaonkar, B., Shenoy, P., Roscoe, T., 2003. Resource overbooking and application profiling in shared hosting platforms. SIGOPS Oper. Syst. Rev. 36 (SI), 239–254. http://dx.doi.org/10.1145/844128.844151.

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

Xavier, M.G., 2019. 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.

Xavier, M.G., Neves, M.V., Rose, C.A.F.D., 2014. A performance comparison of container-based virtualization systems for MapReduce clusters. In: 2014 22nd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing. pp. 299–306. http://dx.doi.org/10.1109/PDP.2014.78.

Xavier, M.G., Rossi, F.D., Rose, C.A.F.D., Calheiros., R.N., Gomes, D.G., 2017. Modeling and simulation of global and sleep states in ACPI-compliant energy-efficient cloud environments. Concurr. Comput.: Pract. Exper. 29 (4), e3839. http://dx.doi.org/10.1002/cpe.3839, e3839 cpe.3839.

Zhang, W., Rajasekaran, S., Wood, T., Zhu, M., 2014. MIMP: Deadline and interference aware scheduling of hadoop virtual machines. In: 2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing. pp. 394–403. http://dx.doi.org/10.1109/CCGrid.2014.101.

Zhang, F., Tang, X., Li, X., Khan, S.U., Li, Z., 2019. Quantifying cloud elasticity with container-based autoscaling. Future Gener. Comput. Syst. 98, 672–681. http://dx.doi.org/10.1016/j.future.2018.09.009.

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

**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.

**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.

**Dionatrã F. 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.

**Cesar 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.