Revista de Informática Teórica e Aplicada - RITA - ISSN 2175-2745
Vol. 29, Num. 03 (2022) 65-75

RESEARCH ARTICLE

Application of Profile Prediction for Proactive Scheduling

Previsão de Perfis de Aplicações para Escalonamento Proativo

Allan Matheus Marques dos Santos[1]*, Raquel Coelho Gomes Pinto[1], Julio Cesar Duarte[1], Bruno Richard Schulze[2]

Abstract: Today, cloud environments are widely used as execution platforms for most applications. In these environments, virtualized applications often share computing resources. Although this increases hardware utilization, resources competition can cause performance degradation, and knowing which applications can run on the same host without causing too much interference is key to a better scheduling and performance. Therefore, it is important to predict the resource consumption profile of applications in their subsequent iterations. This work evaluates the use of machine learning techniques to predict the increase or decrease in computational resources consumption. The prediction models are evaluated through experiments using real and benchmark applications. Finally, we conclude that some models offer significantly better performance when compared to the current trend of resource usage. These models averaged up to 94% on the F_1 metric for this task.
Keywords: application profile – scheduling — cloud computing — intelligent agents

Resumo: Hoje, os ambientes de nuvem são amplamente utilizados como plataformas de execução de grande parte das aplicações. Nesses ambientes, aplicações virtualizadas geralmente compartilham recursos computacionais. Embora isto aumente a utilização do hardware, a competição por recursos pode causar perda de desempenho, e saber quais aplicações podem ser executadas em uma mesma máquina física sem causar muita interferência é a chave para obter um escalonamento mais adequado e um melhor desempenho. Desta forma, é importante prever o perfil de consumo de recursos das aplicações nas suas iterações subsequentes. Este trabalho avalia o uso de técnicas de aprendizado de máquina para a previsão do aumento ou redução no consumo de recursos computacionais. Os modelos de previsão são avaliados através de experimentos utilizando aplicações reais e sintéticas. Por fim, verifica-se que alguns modelos oferecem um desempenho significativamente superior quando comparados com a tendência atual de utilização dos recursos. Tais modelos alcançaram, em média, até 94% em termos de F_1 nesta tarefa.
Palavras-Chave: perfis de aplicações – escalonamento — computação em nuvem — agentes inteligentes

[1] *Military Institute of Engineering (IME), Rio de Janeiro – Rio de Janeiro, Brazil*
[2] *Distributed Computing, National Laboratory of Scientific Computing (LNCC), Petrópolis – Rio de Janeiro, Brazil*
***Corresponding author**: allan.santos@ime.eb.br
DOI: http://dx.doi.org/10.22456/2175-2745.120399 • **Received:** 29/11/2021 • **Accepted:** 20/11/2022

1. Introduction

In recent years, the adoption of cloud computing platforms has rapidly increased. This is true since this paradigm allows better management of on-demand resources for applications and cloud services can be accessed from virtually any computing device connected to the network, such as *smartphones* and Internet of Things (IoT) devices [1].

Besides day-to-day applications, computational clouds are also used for high-performance massively parallel software execution, as is the case in scientific applications [2]. In general, these applications perform complex calculations to simulate or predict phenomena according to a mathematical model. The information obtained by these applications is necessary for scientific research in several fields, such as physics, chemistry and medicine.

Scientific applications may be classified into profiles according to the runtime computational resources consumption [3]. Each profile highlights the type, or set, of resources that the application uses the most. Application profiles can be used to identify application affinity, which indicates how well they can run concurrently without interference. Low affinity indicates a high degree of interference between applications running concurrently, so a high affinity is often desired.

Some applications develop high affinity when executed on the same real machine since the execution of one does not significantly degrade the performance of the others [3].

However, there are also sets of low affinity profiles that, if executed concurrently, can cause interference, affecting application performance when running on the same machine.

This interference phenomenon can be observed in cloud computing environments. In this type of environment, applications from different users are usually isolated in instances of a Virtual Machine (VM) that run concurrently and can be scheduled in the same Physical Machine (PM). Thus, if two low-affinity applications are scheduled in the same PM, performance degradation may occur and, therefore, some applications may take longer to run than expected or services may be unavailable.

Furthermore, in certain contexts such as public clouds, a minimum performance level that the applications need to achieve is usually stipulated. This can be defined in the Service-Level Agreement (SLA), and interference between applications may impact compliance with this agreement. For these reasons, it is important to consider the affinity between applications when running applications concurrently in the cloud. One of the techniques that can be used to mitigate this problem is to predict resources usage by a certain application.

Machine learning techniques, such as neural networks and decision trees, are usually employed to perform predictive tasks on a given phenomenon from a set of historical known data. By applying these techniques in an application affinity environment, it is possible to predict a *profile* for each application based on its resource consumption history. In this way, a dynamic scheduler can be built to avoid, or at least mitigate, interference by preventing virtual machines with low affinity from running concurrently.

The use of virtualization allows application clustering for those that have affinity in the same PM and, at the same time, application isolation for the ones that have low affinity in different PM, thus, maximizing computational resources usage while reducing performance degradation. With this goal in mind, it is necessary that each one of the applications is isolated in a VM and that there is a scheduler capable of using these profile knowledge to control the execution of these VMs in a distributed environment, such as a cloud.

In order to reduce interference in cloud computing, many approaches use scheduling algorithms that take into account application profiles known *a priori*, if any, for their initial allocations. Yokoyama et al.[4] proposed, for example, the estimation of the affinity of applications and then uses this static value to schedule these applications on the most appropriate hosts to avoid interference. Some works also monitor the VM resources consumption of each application to identify profile changes and reallocate VMs through migrations [5].

Affinity-based scheduling models [4] show good results by avoiding running applications with low affinity on the same real machine, for those that always maintain the same profile. Furthermore, with real-time monitoring of virtual machines, it is possible to detect when an application changes its profile and starts to cause, or suffer from, interference [5], allowing a corrective reaction such as the migration of the VM, for instance.

However, by only using *a priori* knowledge profiles and monitoring their changes, it is not possible to anticipate undesirable events and act proactively, in order to optimize application scheduling.

In this context, machine learning techniques allow the development of predictive models that learn applications behavior from data collected through previous execution monitoring. Such models can not only be used to improve the decisions of a virtual environment scheduler, but also allow proactive actions to be taken in undesirable cases.

The objective of this work is to evaluate the use of machine learning techniques to predict changes in application profiles. In this way, it is possible to reduce or even avoid interference between applications with low affinity in computational clouds.

By using the prediction of VM profiles, knowledge of available hardware resources in each PM, and knowledge of the current VM allocation, it is possible to detect when co-allocated VMs may degrade system performance. By predicting these undesirable events, techniques and tools known in the literature can be applied to mitigate, or perhaps eliminate, interference before too much damage is done.

This work is organized as follows. Section 2 presents related works and some comparisons where possible, while Section 3 presents the development of this work. Section 4 presents an analysis of the results obtained with the proposed methodology. Finally, Section 5 presents the conclusion of the work and discusses future works.

2. Related Work

This section presents related work that were used as inspiration for the development of this article. In addition to the main features of each work, comparisons are presented in order to position this work in relation to the others. Table 1 presents a summary of these comparisons.

Shahin[7] presents two algorithms to allow automatic provisioning of new cloud instances using Long Short-Term Memory (LSTM) networks in order to handle request overloads. In the first algorithm, a LSTM model is trained to predict CPU utilization based on historical data that combines normal loads with *slashdot* loads. A *Slashdot* is a sudden ingress of valid traffic and common auto-scaling techniques usually are not able to scale fast enough for this type of workload, which can lead to SLA violations. At runtime, the model is fed with monitoring data and the predictions are passed to a decision-making module. The second algorithm, on the other hand, proposes the use of two prediction models, where the first one is based only on normal load data and the other one is based on *slashdot* loads. At each instant, both models generate a prediction and the associated Mean Absolute Percentage Error (MAPE) is used to decide which prediction is used in the decision making process. Both algorithms reduce the response time compared to other approaches in the literature by up to 300 ms.

Table 1. Summary of related works

Article	Goal	Method	Evaluation
Bernardo, Pinheiro e Pinto[6]	*Slashdot* cloud effects toleration	EMA	Reduction in the effects of *slashdot*
Oliveira et al.[5]	Interference effects reduction	EMA	Reduction in up to 15% of total execution time
Shahin[7]	Automatic scheduling	LSTM	Reduction in average response time of up to 300 ms
Wei et al.[8]	On demand pricing resources	HMM	Increase in profit by about 25%
Guo e Yao[9]	System load prediction	GRU	Smaller MSE and MAPE
Duggan et al.[10]	SLA violations reduction	RNN	85% less violations
This work	Application Profile Prediction	several	up to 94% F_1 score on average

As a source of workload, Shahin[7] used *logs* from HTTP servers provided by NASA [11] (characterized by normal loads) and by a free software installation festival [12] (characterized by *slashdot* loads). The *cloudsim* software is used to simulate the computational cloud environment and the Deeplearning4j library is used as the implementation for the machine learning models.

The analysis of only one resource, the CPU, does not allow a complete characterization of the PMs workload. To improve the prediction of computational resource consumption, Guo e Yao[9] presents a prediction model based on the workload evaluated as a linear combination of CPU, memory, network and disk consumption, where the weights are empirically defined. The authors compared prediction results for several models, including LSTM and Gated Recurrent Units (GRU), using the Mean Square Error (MSE) and MAPE. The error rates obtained by the GRU models were less than half the rates of the other approaches and only the LSTM had a near but larger error. In addition to more accurate results, GRU showed greater efficiency, as its training time was significantly lower, about 71% of the training time of a LSTM model in the same dataset.

To generate the workload data, Guo e Yao[9] monitored the execution of applications for 10 days in an PM, with 8 days reserved for training and 2 days for testing. TensorFlow library was used to implement the models.

Another proposed solution for the *slashdot* problem is PHOENIX [6]. In this work, instead of creating more instances of the application, as is done by Shahin[7], VMs are migrated to balance the system load. The system migrates, at runtime, VMs that present overload to another PM with more available resources, or fixes the overloaded VM and moves the others, isolating the overload. The system also applies two types of load balancing: proactive, when there is no overload; and reactive, when an overload is detected. For overload detection, PHOENIX considers CPU, memory and network usage, but does not apply machine learning techniques. The overload detection algorithm is based on trend analysis with an exponential moving average while using the SLA fulfillment to evaluate the adopted approach.

In computational clouds, VMs migration is a widely used technique to reallocate virtualized applications. One of the reasons for relocating applications is to reduce performance loss due to excessive concurrency, evidenced by low affinity between applications.

Oliveira et al.[5] presents a scheduler capable of identifying low affinity between applications running on the same PM and migrating one of the applications to another machine, freeing resources and interrupting interference. The system uses empirically defined consumption thresholds and an Exponential Moving Average (EMA) to identify trends in consumption that can detect changes in application profiles at runtime. The VMs are non-intrusively monitored and, when applications with low affinity profiles are identified as running concurrently, one of the VMs is migrated to another PM.

The architecture proposed by Oliveira et al.[5] consists of several monitors (one for each application) and a central scheduler. These monitors collect application resource usage data while assessing changes in consumption profile, notifying the scheduler of detected changes. The scheduler then checks whether the change can interfere with other applications, and, if this happens, it migrate some VMs to a more suitable host in the cloud environment.

Wei et al.[8] presents a solution for scheduling resources in a Software as a Service (SaaS) computational cloud environment that offers resources for other Platform as a Service (PaaS) and SaaS clouds. In this proposed model, the price of each infrastructure resource varies, not only with consumption, but also with demand. The authors consider the price of infrastructure provider resources, and consumers and service providers needs. A Hidden Markov Model (HMM) model is used to predict the demands for each resource, based on the client's consumption history obtained through experiments. By using this information, the prices of the offered resources are then evaluated.

The behavior of the service providers competing for resources can then be modeled using tools applied in the financial market study. The authors use a model based on game theory, called Imperfect Information Stackelberg Game Model (IISG) to find a balance situation, that is, the set of all actions taken individually by each service provider that best meets the needs of the service providers, minimizing resource prices.

Several works seek to predict time series, in particular, the consumption of computational resources presenting learning models trained to predict the next element. However, tasks that involve migrating VMs, creating new instances of applications

and provisioning resources on demand in the cloud can take several minutes. So it is necessary to make this prediction with greater anticipation. Duggan et al.[10] compares the use of various machine learning techniques, including a Recurrent Neural Network (RNN), for predicting CPU consumption and network bandwidth.

The experiments presented demonstrate that recurrent networks offer better performance. In addition, simulations were performed based on application execution histories in a cloud environment. In these simulations [10], it was possible to anticipate the moment to migrate the VMs and, thus, reduce SLA violations.

Duggan et al.[10] obtains the network consumption dataset from a study conducted in the Amazon EC2 cloud, consisting of monitoring a period of 1 day. A larger dataset is generated, considering the Gaussian distribution of the data, then, CPU data is taken from the Google *cluster data trace* dataset, which is composed of a set of logs usage of about 12,000 machines in a cluster over a period of 29 days. The cloud environment is simulated using Cloudsim software to get an estimate of migration times.

Table 1 presents the main characteristics of each of the works detailed in this Section. We can see that the main difference from the other works is that, in this work, machine learning techniques are used to estimate the future affinity between applications based on the prediction of resource consumption.

3. Application Profile Prediction Models

We can define a computing environment as a set of physical machines P, and a set of virtual machines V. Each virtual machine runs an instance of some application from its suite of applications A and each real machine runs a subset of the virtual machines. For practical purposes, in this work, a virtual machine is treated as an application instance since the cardinality of the relationship between these entities is always 1:1.

Let a_i and a_j be applications with low affinity profiles running on VMs v_i and v_j, respectively, which are running on the same PM p_i. In this case, the performance of p_i will be degraded, increasing the execution time of the tasks of this PM, not just v_i and v_j. If their profiles are static, then it is possible to develop a scheduler that avoids this situation by not allocating, as far as possible, two VMs with low affinity profiles in the same PM. Nonetheless, this is often empirically verified not to be true.

As resource consumption profiles change over runtime, so does affinity. Therefore, an initial scheduling that minimizes interference, considering a static profile, is not enough to maintain interference low throughout the entire run. Also, monitoring the consumption of VM resources while running, only allows you to perform reactive actions to control this situation.

Our approach consists of using machine learning algorithms in order to predict the computational resources consumption profile of applications based on their monitoring, thus, acting proactively. We chose the architecture proposed by Oliveira et al.[5] to be the basis of our work because it allows the addition of new strategies for monitoring and identifying profile changes. Figure 1 illustrates the general architecture of the scheduling system with a central scheduler and multiple pairs of VMs and monitors.

The applications run on the VMs while the monitors collect and process data about their resource consumption. From this data, the monitors send notifications to a central scheduler whenever a profile change is predicted. The scheduler can then observe the profiles of the other VMs as well as the environment, and thus, if profiles with low affinity are predicted to be on the same host, it can decide to migrate some VMs to avoid or reduce interference.

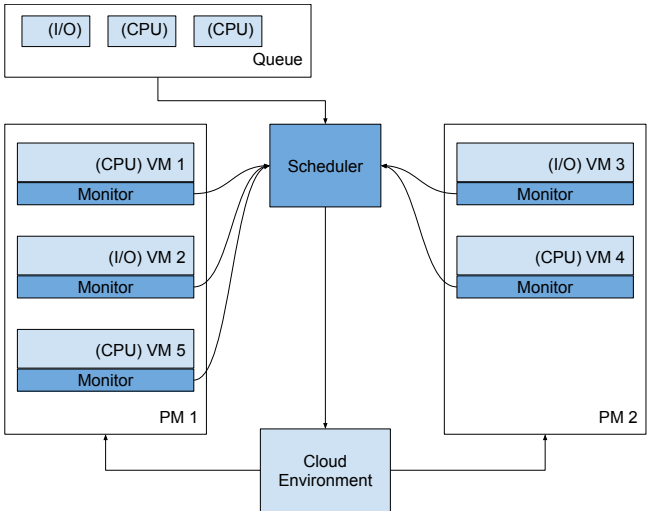


Figure 1. Scheduling system architecture based on the base architecture proposed by Oliveira et al.[5].

Figure 2 illustrates data flow for an application in the cloud environment. The *Data Collector* module is responsible for monitoring the VM, extracting `CPU` and `I/O` (disk) usage data. This component plays an important role both in the training phase of the machine learning algorithms and in the application of their derived classifiers, as it generates the necessary raw attributes.

In order to train the classifiers that compose the *Predictor* module, these generated attributes are evaluated in an attribute engineering phase, to generate derived attributes that capture the current trend and future states. To capture the current trend, we create attributes with the difference in consumption between consecutive steps of time. For the future states, we create binary attributes that are the classification targets. These attributes are true (1) if the future consumption is greater than or equal to the current consumption and false (0), otherwise.

Furthermore, a search for the best hyper-parameters is performed, with the objective of comparing different versions

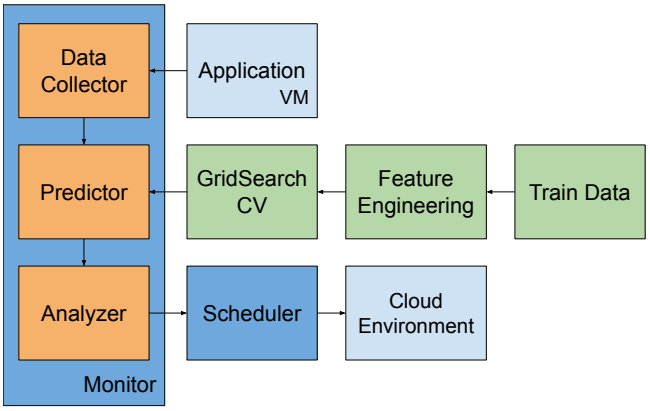


Figure 2. Data flow architecture for a single VM

of the classifiers derived from the same algorithms, while establishing the best setups for each profile. Each hyper-parameter set that generates the classifier with the best results in a validation subset of the training set is selected for the final training with the full set.

The *Analyzer* module, on the other hand, implements heuristics that can decide whether the trend predictions made by the *Predictor* module consist of a likely profile switch. Whenever its decision is true for the profile switching, a notification is triggered for the *Scheduler* module. Finally, the *Scheduler* module uses APIs provided by the *Cloud Environment* to request a migration of the VMs with predicted low affinity profiles to more suitable PMs.

The cloud environment used in the experiments is OpenStack with a KVM virtualization system, and the virtual machines are configured with 4 CPU cores, 5GB of RAM and 20GB of disk each. Due to the KVM memory ballooning driver, it is impractical to observe the real variation of memory consumption in the guest with the implemented monitor. This is due to the monitor treating each guest as a host process, and, thus, it is only possible to know when consumption increases [5, 13] and, due to this, only `CPU` and `I/O` consumption are considered.

In order to have a binary class, an attribute *target* is derived consisting of a boolean value $x_i \geq x_{i+n}$, indicating if the current consumption value of a given resource x will increase or remain constant in n time steps in the future. Differences between the last k observations $(x_i - x_{i-k})$ are also added to aid trend predictions.

The prediction models receive, as input, the current value and the latest differences (current trend) of all resources, and its task is to predict whether or not it will increase or stabilize in n steps in the future for a specific resource. For each value of n, a different model is trained. The values used in our experiments for n are 1, 20, 100 and 200 time steps, indicating predictions in the near and distant future. As the interval between observations is 3 seconds per time step, those predictions translate for 3 seconds, 1, 5, and 10 minutes in the future, respectively.

For each type of application, type of computational resource, and step in time, a prediction model is trained that uses a hyper-parameter (*grid search*) with a stratified *k-fold* validation scheme in relation to the target and randomized attribute. The *scikit-learn* software library [14] was used, as it offers efficient implementations of the chosen algorithms. These choices were made based on the success in other similar tasks, and, in order to allow a comparison with other works not totally related. So the following machine learning algorithms are used for the prediction models.

- *Gaussian Naive Bayes* (GNB): it evaluates the conditional probability of the classes given the features [15].
- *AdaBoost Classifier* (ABC): it trains a classifier in the training dataset and then trains copies of that classifier, adjusting weights according to the performance of the previous ones. Therefore, it can improve performance of difficult instances. The implementation of the meta estimator used in this work is presented in [16].
- *Random Forest Classifier* (RFC): it is a meta-estimator that combines decision trees. Each tree is trained with a subset of the training data, thus, it can improve performance and control over-fitting [17].
- *Multi-Layer Perceptron Classifier* (MLPC): learns a representation of data using a backpropagation algorithm [18]. It can learn non-linear relationships.
- *Support Vector Machine Classifier* (SVC): searches for the hyper-plane that separates the points by classes with the larger possible margin. It then uses the hyper-plane to predict a class for new points. The implementation used in this work is described in [19].

In order to better evaluate the trained algorithms, a simple *baseline* model is also used, which consists of replicating the value of the previous trending observation. In other words, if, in the last iteration, the trend indicates increased consumption, the prediction will be increase (y=1) for the next steps, otherwise, it predicts no increase (y=0).

The application monitor collects resource usage data at each time step and is capable of issuing profile switch notifications. Oliveira et al.[5] proposed a non-predictive monitoring algorithm that issues notifications only after profile change. Upon receiving this message, the scheduler will check if the profile change implies low affinity among all tasks from the same host, and, if this occurs, the scheduler will decide to migrate the task to a more suitable *host*. Algorithm 1 presents an overview of the decision process adopted by Oliveira et al.[5].

By incorporating the machine learning models, it is possible to predict the application profile by combining the individual predictions. Algorithm 2 is the responsible for that, treating the predictions as a weighted vote, where the weights higher for short-term forecasts.

Algorithm 1: Profile Change Scheduling

```
input  : task, host, profile
output : migrates if profile change causes low affinity
affinity ← calc_affinity (task, host, profile) ;
if affinity < 0.6 then
    new_host ← choose_best_host (task,
      profile) ;
    migrate (task, host, new_host) ;
end
```

Algorithm 2: Monitoring with Prediction

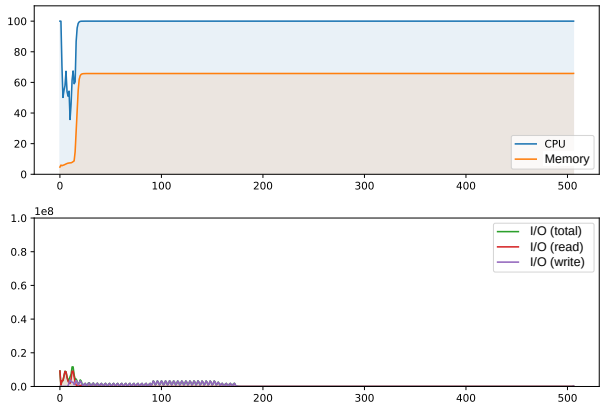
```
input  : VM, Weights, Resources, Window
output : profile change notification
history ← List(size = Window);
profile ← null;
while is_running (VM) do
    usage ← collect_usage (VM, Resources) ;
    /* keep only the last Window
       values                                */
    history ← append (history, usage) ;
    for r in Resources do
        y_pred ← predict (history, r) ;
        votes ← sum(Weights_r * y_pred);
        if votes > (sum(Weights_r)/2) and
          profile ≠ r then
            profile ← r;
            notify (profile);
        end
    end
end
```

4. Results and Discussion

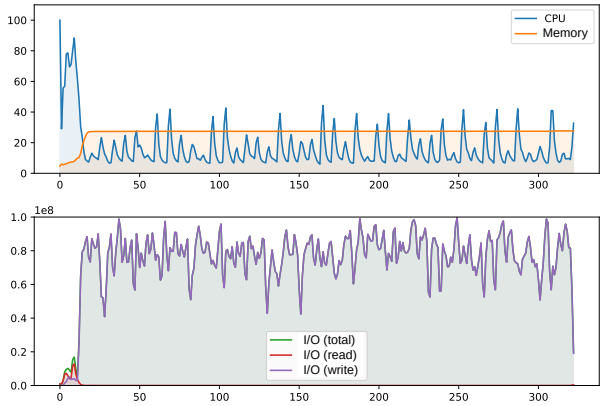
This section presents an evaluation of the presented modeling in the context of its application for the prediction of the following time steps, 1, 20, 100, 200, which, in our case, are, respectively, 3 seconds, and 1, 5, and 10 minutes, respectively. In these tests, we compare all proposed machine learning algorithms for prediction and the baseline presented in Section 3.

The chosen applications used for the experiments were Blast, Montage, IOzone and HPL. Blast and Montage are real applications, while the other two are benchmarks. Blast is a bioinformatics software used to search for similarities between biological sequences like proteins in a database [20]. Montage is an astronomy application toolkit [21] that allows the composition of satellite image mosaics. HPL, on the other hand, is a benchmark that solves dense linear equations systems in double precision (64 bit) [22]. HPL is usually used to estimate the performance of distributed computing environments such as clusters. Finally, IOzone is a tool to test the speed of the file system [23]. It generates read and write loads with various size settings and then measures the performance of the storage system. In this paper, we use IOzone to simulate an IO-bound application.

We ran several iterations of these applications in a cloud environment while monitoring their resource consumption. So, with the data monitoring, it was also possible to generalize the applications observed behavior into classes of application. For instance, the HPL application mainly displays a CPU usage profile, as shown in Figure 3a. On the other hand, the application IOzone mainly presents an I/O usage profile, as shown in Figure 3b.



(a) HPL monitoring sample



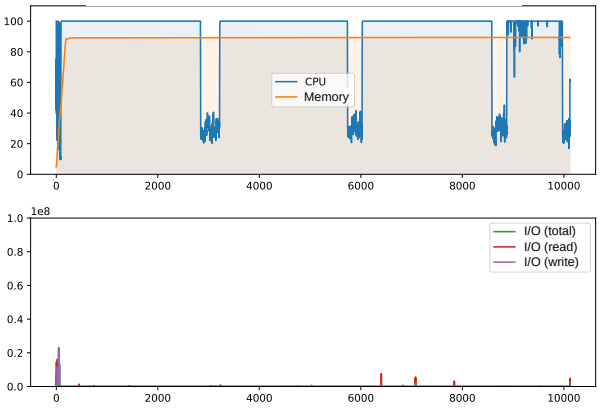
(b) IOzone monitoring sample

Figure 3. Synthetic application sample monitoring

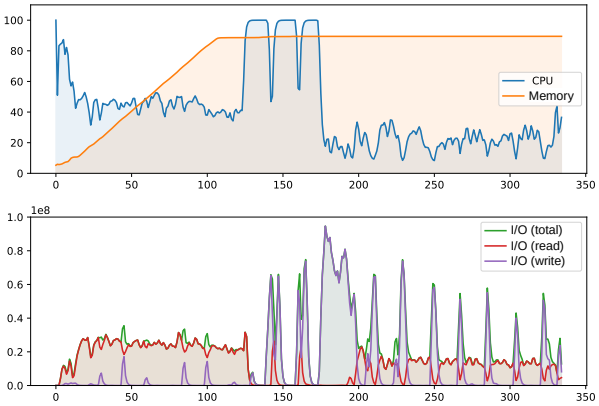
The Blast application has an intensive CPU usage profile as well as HPL, however it is possible to notice in Figure 4a that memory consumption RAM is also high and, therefore, it is also possible to classify this application as Mem profile usage.

Montage application exhibits more varied behavior over time. This is due to the fact that this application runs a *workflow* in which each step may present a different profile. Figure 4b illustrates one monitoring of this application in which it is possible to see how this application changes from a more intensive CPU consumption to a more intensive I/O consumption throughout its execution.

Resource consumption monitoring data from the applications was then used as a time series dataset for the machine



(a) `Blast` monitoring sample



(b) `Montage` monitoring sample

Figure 4. Real-world application sample monitoring

learning models. In each execution, we diversified the available application parameters so that the dataset to represent a global application profile and not a single run with a specific set of parameters.

In the vast majority of cases, the performance of those models surpasses the one provided by the baseline and, as expected, deteriorates as we increase the time step, i.e., we predict more in the future. Furthermore, in general, the ensemble methods, ABC and RFC, outperform the others algorithms by a small advantage. MLPC and SVC present a reasonable performance compared to the others, with emphasis on the MLPC when applied to `CPU` usage prediction for Blast. GNB, on the other hand, presents, in general, the worst results and also some unexpected behaviors. This behaviour is expected since GNB assumes statistical independence among the used attributes, which in our case, is not true.

4.1 Class Balancing

The proposed built target attributes for the binary classification process, $y = 1$, which represents if consumption will be greater or equal, in k time steps, and $y = 0$, otherwise, presented an imbalance characteristic in terms of example amounts. So, for certain training instances, there are many more examples for one class than for the other. For example, in the case of `CPU` usage prediction for the HPL application at 1 time step in the future, there are only 191 examples with class ($y = 0$) while for class ($y = 1$) there are 4658 examples. Table 2 presents the class amounts for each case.

App	Steps	CPU (y=0)	CPU (y=1)	I/O (y=0)	I/O (y=1)
Blast	1	14925	69852	9651	75126
	20	19237	65540	9829	74948
	100	21839	62938	11973	72804
	200	24675	60102	13881	70896
Montage	1	2870	4147	3800	3217
	20	3588	3429	3838	3179
	100	4497	2520	4832	2185
	200	5750	1267	5582	1435
HPL	1	191	4658	1574	3275
	20	578	4271	2030	2819
	100	1679	3170	2941	1908
	200	2479	2370	3776	1073
IOzone	1	7921	4707	6130	6498
	20	6725	5903	6408	6220
	100	7937	4691	7299	5329
	200	9179	3449	8424	4204

Table 2. Number of examples of each class for each application and steps.

Due to class imbalance, to quantify the performance evaluation of the provided models, the F_1 metric was used [24]. F_1 is defined as the harmonic mean between precision and recall, and can be expressed by the following equation:

$$F_1 = 2 \cdot \frac{precision * recall}{precision + recall}.$$

Where precision and recall are information retrieval performance metrics that allow evaluating a classifier from different perspectives. Precision measures the number of correct values retrieved from a specific class out of all class values that were retrieved, while recall measures the number of correct values out of all possible values for that class.

Figures 5 and 6 present the performance of the models for the Montage application while Figures 7 and 8 present the models for the Blast application. On average, the results for F_1 in the trend prediction task for consumption resource usage had values of up to 94% score.

Among the real-world applications used, the models showed better performance for Blast, both in `CPU` prediction and in `I/O`. This is probably due to Blast not having much variation in its consumption profile. On the other hand, Montage, which has several stages with different characteristics in its workflow, showed the worst results.

The predictive models for HPL present better performance when compared to the ones for IOzone, both benchmark applications. We believe this is affected by the greater variation in terms of consumption by IOzone. An interesting behavior here is that GNB has a better performance when compared to

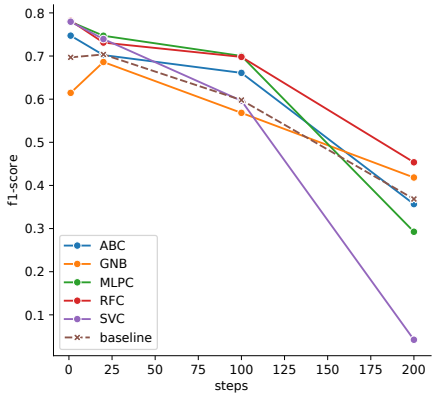


Figure 5. F_1-score for `CPU` prediction in Montage

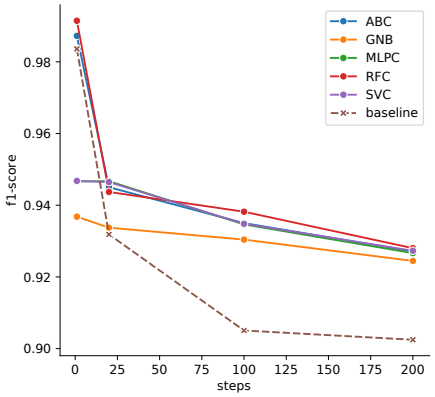


Figure 8. F_1-score for `I/O` prediction in Blast

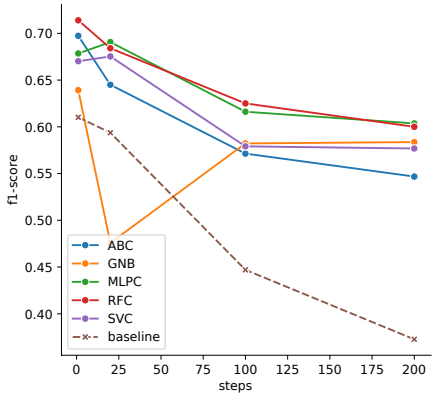


Figure 6. F_1-score for `I/O` prediction in Montage

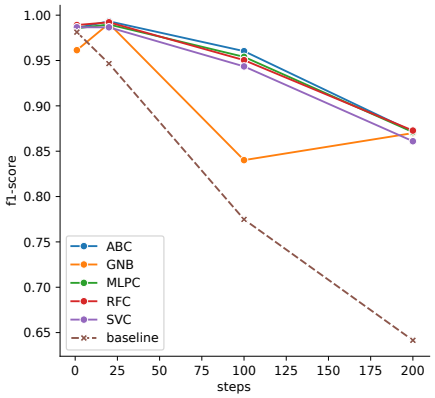


Figure 9. F_1-score for `CPU` prediction in HPL

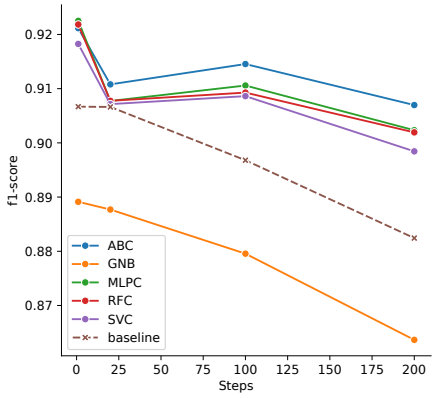


Figure 7. F_1-score for `CPU` prediction in Blast

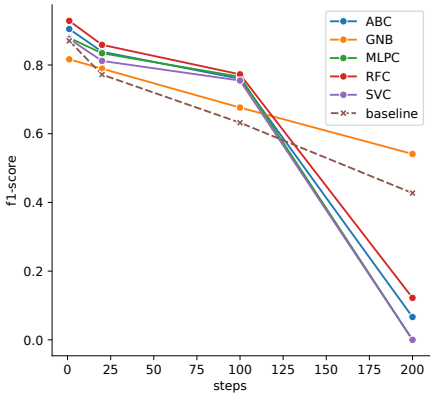


Figure 10. F_1-score for `I/O` prediction in HPL

the other algorithms when predicting IO for time step 200, in both applications. Figures 9 and 10 present the models for the HPL application while Figures 11 and 12 present the same models for the IOzone application.

Table 3 presents the "best" algorithms for each discussed application class. For `CPU` usage prediction in the `CPU` class, the recommended algorithm is ABC, which achieved the best performance for time steps 20 and 100, while having slightly small performance in time steps 1 and 200. As for `I/O` prediction, the algorithm RFC presented the best results in 3 of the 4 time steps analyzed. Nevertheless, for time step 200, it was outperformed by the *baseline* algorithm.

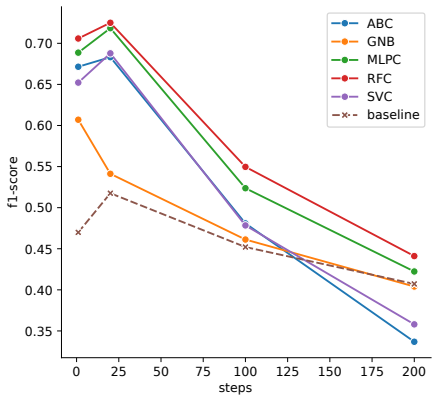


Figure 11. F_1-score for `CPU` prediction in IOzone

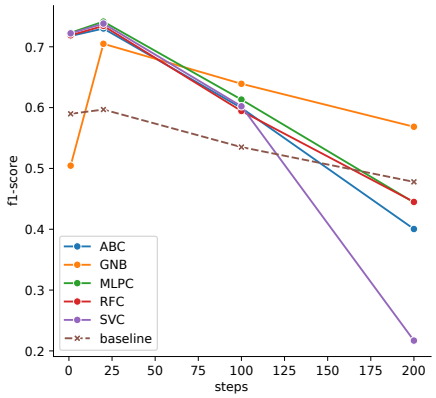


Figure 12. F_1-score for `I/O` prediction in IOzone

For the `I/O` class, the recommended algorithm in terms of predicting `CPU` consumption is RFC, which achieved the best performance for all time steps for this class. For `CPU` prediction, the algorithm GNB was chosen because it achieved the best performance in time steps 100 and 200, while slightly being outperformed for time step 20 and being surpassed by the *baseline* in step 1.

For class Mem, the best algorithm was ABC in terms of `CPU` prediction. This algorithm achieved the best performance in 3 of the 4 time steps analyzed, being only outperformed in time step 1 by a small difference. For `I/O` prediction, the chosen algorithm is RFC, which also achieved the best performance in 3 of the 4 time steps, being only outperformed in time step 20 by a small margin of difference.

Finally, for the `Workflow` class, the algorithm RFC is recommended for both `CPU` and `I/O` consumption prediction. This algorithm achieved the best performance for two time steps in each prediction type while not being outperformed by the baseline in any of the cases.

As only one application was selected for each class, the chosen algorithms may not present high performance for new applications. Keeping this in mind, it is important to maintain

Application Class	Application	`CPU`	`I/O`
CPU	HPL	ABC	RFC*
E/S	IOzone	RFC	GNB*
Mem	Blast	ABC	RFC
Workflow	Montage	RFC	RFC

Table 3. Best algorithms for each class regardless of time step. Algorithms marked (*) underperformed *baseline* at some time step

data collection for new applications, while also training new models to reflect new applications behaviour.

In the case of new applications, two strategies can be adopted. If the application's class information is unknown, Oliveira et al.[5]'s approach can be used to identify profile changes while the application's consumption history is collected, which can then be used to train a new specific model. Secondly, it is possible to apply one of the class prediction models assuming that the behavior of the new application is similar to any of the applications already used for this class. Likewise, in this case, also data collection is important to update the model with this possible new behaviour. In both cases, after collecting a sufficient sample amount, a specific model for the new application can be trained in order to be used to predict consumption increase in future executions. Also, the class prediction model, if known, can be updated, including the consumption history data of this new application in the training.

As presented in the results provided in this section, the use of machine learning algorithms has the ability to reliably predict the increase in resource consumption by virtualized applications up to several minutes into the future, especially when compared to simple heuristics. By using this strategy, cloud application scaling systems can act early to better mitigate performance loss due to interference.

5. Conclusions

Interference caused by resources competition can cause significant performance loss in virtualized applications in cloud environments. Many application schedulers in these environments try to reduce the effects of this problem by monitoring application consumption and taking actions when interference starts.

However, waiting for interference to start can be too late to mitigate its effects, and predict a change in the profile application in advance would allow the VM migration to begin before the performance drop starts.

In this work, we evaluate the use of machine learning algorithms to predict the increase in resource consumption, enabling the detection of profile changes in applications running in a virtualized environment. Furthermore, we present an scheduling architecture (Figures 1 e 2) that combines existing architectures with machine learning techniques that can be used to develop a proactive scheduler that acts in advance to avoid the loss of performance by interference.

We can conclude that these techniques allow us to predict whether consumption will increase for up to several minutes in the future. In our experiments, we observed up to 94% F_1 score on average, in terms of capturing the future trend of resources usage.

Likewise, it is expected that the present work can be inserted into a larger system that allows carrying out the entire end-to-end decision process regarding the migration of virtualized applications. In this context, it is expected to be able to further explore the architectures developed here in possible future works where a proactive dynamic scheduler can be implemented with the development of new heuristics to identify the applications profile, and consequently, profile changes based on their consumption predictions.

Furthermore, it is possible to generalize application prediction models, grouping them into classes according to applications behavior. This facilitates the applications handling, for known classes, that do not have a trained model. If neither the application nor its class are known, it would still be possible to act, applying the base algorithm proposed at Oliveira et al.[5]. In all these cases, the dataset is incremented with the application's resource consumption history, allowing the training of new models.

Future work also include exploring the use of other machine learning algorithms, increasing the hyperparameters search space when training the models, predicting consumption values with regression tasks to complement the trend prediction that was presented in this work and the study of new applications to improve predictive models for application classes.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Finance Code 001.

Author contributions

All authors have contributed equally to the development of the methodology proposed in this work. Allan Santos was responsible for implementing the machine learning experiments.

References

[1] RAY, P. P. A survey of iot cloud platforms. *Future Computing and Informatics Journal*, Elsevier, [S. l.], v. 1, n. 1-2, p. 35–46, 2016.

[2] GARG, S. K. et al. Environment-conscious scheduling of hpc applications on distributed cloud-oriented data centers. *Journal of Parallel and Distributed Computing*, Elsevier, [S. l.], v. 71, n. 6, p. 732–749, 2011.

[3] MURY, A. R. et al. A concurrency mitigation proposal for sharing environments: An affinity approach based on applications classes. In: SPRINGER. *International Conference on Intelligent Cloud Computing*. [S.l.], 2014. p. 26–45. Disponível em: ⟨https://link.springer.com/chapter/10.1007/978-3-319-19848-4_3⟩. Acesso em: 25 set. de 2018.

[4] YOKOYAMA, D. et al. Affinity aware scheduling model of cluster nodes in private clouds. *Journal of Network and Computer Applications*, v. 95, p. 94 – 104, 2017. Disponível em: ⟨http://www.sciencedirect.com/science/article/pii/S1084804517302497⟩. Acesso em: 25 set. de 2018.

[5] OLIVEIRA, V. et al. Alocação de ambientes virtuais com base na afinidade entre perfis de aplicações massivamente paralelas e distribuída. *Brazilian Journal of Development*, [S. l.], v. 5, n. 10, p. 21905–21925, 2019.

[6] BERNARDO, E. H. C.; PINHEIRO, W. A.; PINTO, R. C. G. Phoenix: A framework to support transient overloads on cloud computing environments. *International Journal of Information Technology and Computer Science*, [S. l.], v. 10, p. 33–44, 2018.

[7] SHAHIN, A. A. Automatic cloud resource scaling algorithm based on long short-term memory recurrent neural network. *International Journal of Advanced Computer Science and Applications*, The Science and Information Organization, [S. l], v. 7, n. 12, 2016. Disponível em: ⟨http://dx.doi.org/10.14569/IJACSA.2016.071236⟩. Acesso em: 30 out. de 2018.

[8] WEI, W. et al. Imperfect information dynamic stackelberg game based resource allocation using hidden markov for cloud computing. *IEEE Transactions on Services Computing*, IEEE Computer Society, Los Alamitos, CA, EUA, v. 11, n. 1, p. 78–89, 2016. Disponível em: ⟨http://or.nsfc.gov.cn/bitstream/00001903-5/481218/1/9913227433.pdf⟩. Acesso em: 31 out. de 2018.

[9] GUO, Y.; YAO, W. Applying gated recurrent units pproaches for workload prediction. In: IEEE. *NOMS 2018-2018 IEEE/IFIP Network Operations and Management Symposium*. Taipei, Taiwan, 2018. p. 1–6.

[10] DUGGAN, M. et al. A multitime-steps-ahead prediction approach for scheduling live migration in cloud data centers. *Software: Practice and Experience*, Wiley Online Library, [S. l.], v. 49, n. 4, p. 1–23, 2018.

[11] NASA. *NASA-HTTP - Two Months of HTTP Logs from the KSC-NASA WWW Server*. 2016. Disponível em: ⟨http://ita.ee.lbl.gov/html/contrib/NASA-HTTP.html⟩. Acesso em: 27 nov. de 2018.

[12] NEWTON, M. *The Slashdot Effect*. 2016. Disponível em: ⟨http://slash.dotat.org/~newton/installpics/slashdot-effect.html⟩. Acesso em: 27 nov. de 2018.

[13] GUO, F. *Understanding memory resource management in vmware vsphere 5.0*. [S.l.]: Vmware, 2011.

[14] PEDREGOSA, F. et al. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, [S. l.], v. 12, p. 2825–2830, 2011.

[15] ZHANG, H. The optimality of naive bayes. *The Florida AI Research Society*, Flórida, EUA, v. 1, n. 2, p. 3, 2004.

[16] HASTIE, T. et al. Multi-class adaboost. *Statistics and its Interface*, International Press of Boston, EUA, v. 2, n. 3, p. 349–360, 2009.

[17] BREIMAN, L. Random forests. *Machine learning*, Springer, [S. l.], v. 45, n. 1, p. 5–32, 2001.

[18] RUMELHART, D. E.; HINTON, G. E.; WILLIAMS, R. J. Learning representations by back-propagating errors. *Nature*, Nature Publishing Group, [S. l.], v. 323, n. 6088, p. 533–536, 1986.

[19] CHANG, C.-C.; LIN, C.-J. Libsvm: a library for support vector machines. *ACM transactions on intelligent systems and technology (TIST)*, Association for Computing Machinery, Nova York, EUA, v. 2, n. 3, p. 1–27, 2011.

[20] ALTSCHUL, S. F. et al. Basic local alignment search tool. *Journal of molecular biology*, Elsevier, [S. l.], v. 215, n. 3, p. 403–410, 1990.

[21] JACOB, J. C. et al. Montage: An astronomical image mosaicking toolkit. *Astrophysics Source Code Library*, [S. l.], v. 1, p. 10036, 2010.

[22] DONGARRA, J. J.; LUSZCZEK, P.; PETITET, A. The linpack benchmark: past, present and future. *Concurrency and Computation: practice and experience*, Wiley Online Library, United Kingdom, v. 15, n. 9, p. 803–820, 2003.

[23] POLTE, M.; SIMSA, J.; GIBSON, G. Comparing performance of solid state devices and mechanical disks. In: IEEE. *2008 3rd Petascale Data Storage Workshop*. Austin, Texas, EUA, 2008. p. 1–7.

[24] BRANCO, P.; TORGO, L.; RIBEIRO, R. P. A survey of predictive modeling on imbalanced domains. *ACM Computing Surveys (CSUR)*, Association for Computing Machinery, Nova York, EUA, v. 49, n. 2, p. 1–50, 2016.