Using Machine Learning Techniques to Classify the Interference of HPC applications in Virtual Machines with Uncertain Data

Rafaela C. Brum[1], **Flavia Bernardini**[1], **Maicon Melo Alves**[2], **Lúcia Maria de A. Drummond**[1]

[1]Instituto de Computação
Universidade Federal Fluminense (UFF) – Niterói, RJ – Brasil

[2]PETROBRAS – Macaé, RJ – Brasil

rafaelabrum@id.uff.br, {fcbernardini, mmelo, lucia}@ic.uff.br

Abstract. *This work aims to predict the level of interference caused by concurrent access to shared resources, such as cache and main memory, that can drastically affect the performance of HPC applications executed in clouds, by using some well-known machine learning techniques. As the user does not know the exact number of resource accesses in practice, we propose a human-readable categorization of these accesses. The used dataset contains information about synthetic and real HPC applications, and, to reflect the uncertainty of the user categorization, we inserted some noisy data in it. Our results showed that our approach could correctly predict the level of interference in most cases, indicating that it can be a practical solution.*

1. Introduction

Nowadays, cloud computing has become a feasible and interesting option to run HPC (High-Performance Computing) applications. This computational paradigm provides valuable benefits such as rapid provisioning of resources and a significant reduction in operating costs related to energy, software license, and hardware obsolescence. What was once seen as a potential option to execute these applications, the cloud has now proved to be a safe, reliable, and affordable alternative to perform this sort of computation. Due to these advantages, the execution of HPC applications has moved from dedicated infrastructures to the computational environment offered by cloud providers more frequently.

Although HPC applications can satisfactorily execute in cloud environments, they can experience severe performance degradation caused by a cross-application interference problem. This problem originates from resource sharing policies commonly employed by cloud providers, where one physical server can host many virtual machines holding distinct applications. In this scenario, applications allocated to the same physical resource, even isolated in their virtual machines, may contend for shared and non-sliceable resources like cache and main memory. Consequently, this dispute over shared resources can lead to a drastic reduction in co-located applications' performance.

To tackle this problem, some works such as [Alves and de Assumpção Drummond 2017, Ludwig et al. 2019, Ren et al. 2019, Zacarias et al. 2019, Meyer et al. 2020] proposed models to predict the level of interference suffered by a set of applications allocated to the same environment. Although those

proposed models presented satisfactory results, some rely on normalized application access rates to shared resources, which are hard information to obtain, especially for end-users. Even though using categorical input features, others did not evaluate the negative impact that uncertain values, provided by end-users, could produce on prediction models. Therefore, it is worth to mention that this requirement for a quite precise and non-human readable information can lead the prediction model to make serious mistakes, besides preventing it from being used in practice by a real cloud provider.

In this paper, we propose a human-readable categorization of the application access to each shared resource, besides taking into account a level of imprecision of input information provided by application users. This scale, based on the Likert one [Likert 1932], is usually employed in research questionnaires and gives some score to each answer in an ordinary way. Categorization is usable in practice as the user does not know the exact number of accesses to each resource and does not execute extra tests only to obtain those numbers. However, this categorization can lead to imprecision, for example, when there is some uncertainty in the application behavior, and the user has to guess it. So, we inserted some noisy data in our dataset. The created dataset contains data from synthetic and real HPC co-located applications in different combinations of the number of processes and virtual machines. To predict how much the co-located applications' performance degraded, i.e. the level of interference, in this new categorized dataset, we used the following well-known machine learning techniques: K-Nearest Neighbours (K-NN), Random Forest, Naive Bayes and Support Vector Machine (SVM).

We created two groups of datasets representing the different number of co-located applications. The first group represents two co-located applications within 226 data objects. The other group of datasets contains three co-located applications' data with 385 objects. Our results showed that the tested classifiers could correctly predict the level of interference in up to 80.09% of the dataset with two co-located applications. When we tested the datasets with three co-located applications, the classifiers can predict the level of interference correctly in up to 95.58% of the dataset.

The work has the following contributions: (i) an interference dataset based on a 5-scale categorization of the access to the shared resources: the shared last level cache (SLLC), the main memory (DRAM), and virtual network (NET); and (ii) a comparison of the machine learning techniques, K-Nearest Neighbours (K-NN), Random Forest, Naive Bayes and Support Vector Machine (SVM), in their ability to deal with uncertain data to predict the interference level of co-located applications.

The remainder of this paper is organized as follows. Section 2 presents the main Machine Learning definitions and notations used in this work. Section 3 presents some related works. Section 4 presents our created datasets and section 5 presents our experimental results. Section 6 presents our conclusions and future work.

2. Machine Learning Definitions and Notations

In this work, we are focused on supervised learning algorithms, which expect as input a training dataset and outputs an estimator of the class for a new object. A training dataset S is a set of N classified objects $\{(\mathbf{x}_1, y_1), ..., (\mathbf{x}_N, y_N)\}$, chosen from a domain X with an arbitrary, fixed and unknown distribution $\mathcal{D}$. These true classification values are given by some unknown function $y = f(\mathbf{x})$. The $\mathbf{x}_i$ objects are typically vectors of the form

$(x_{i1}, x_{i2}, ..., x_{im})$, whose values can be discrete or real. Each value x_{ij} denotes the value of the j-th feature X_j of the object $\mathbf{x}_i$. In this work, our proposal models the output feature as five levels of interference between co-located applications, which leads to a type of classification problem. Given a set T of labeled training examples for classification purposes, the learning algorithm induces a *classifier* $\mathbf{h}(\mathbf{x})$, which is a hypothesis about the true unknown function f. Given new $\mathbf{x}$ values, $\mathbf{h}(\mathbf{x})$ predicts the corresponding y value.

Our input features can be represented as numerical classes and are independent of each other. So, we used the following classifiers in our experiments: K-Nearest Neighbours (K-NN), Random Forest (RF), Naive Bayes (NB), and Support Vector Machines (SVMs). K-NN [Cover and Hart 1967] classifiers use a distance metric to classify a new object. Each new object calculates the distance between this one to all the objects in the database and uses the K nearest objects to decide which label the new object belongs to. RF [Breiman 2001] trains many decision trees for each dataset, considering different random seeds. The final classification of a new object comes from the label voting of all trees. NB is a probabilistic model [P. Domingos and Pazzani 1997], constructed by assuming that each feature X_i is independent of the others. This classifier uses the *a priori* probabilities of all pairs of feature values and labels, calculated using the training dataset, to predict the label of a new object. SVMs are based on the statistical learning theory [Cristianini and Shawe-Taylor 2000]. They are capable of obtaining a classifier with high generalization capacity. SVM was proposed to be a binary classifier (*i.e.*, a classifier constructed for a dataset with only two labels) that finds a frontier between the objects of each label, using some of them as an acceptable margin. The frontier design can be linear or based on a kernel passed to the model in the training phase. When there are more than two labels in the available dataset, the dataset must be decomposed into various datasets for constructing binary SVMs. There are two approaches for decomposing the dataset: The 'one-versus-others' and the 'one-versus-one' approaches. 'One-versus-others' fix one class as a positive class, and all the others are the same negative class, constructing L datasets. In our case, five datasets are constructed, one for each class being the positive, and so five SVMs are constructed. 'One-versus-one' considers the combination of all pairs of labels. So, there are $L \times (L-1)/2$ pairs of labels. One dataset is constructed for each pair of classes, and only objects labeled with one of the classes in the pair belong to the dataset. In our case, as there are five classes, this approach constructs ten datasets, and so ten SVMs are constructed. The former approach has the advantage of constructing fewer classifiers, but the decomposition may generate unbalanced datasets, *i.e.*, datasets with at least one label labeling a low proportion of objects. The later has the advantage of more probably generating balanced datasets, although it can lead to the construction of many classifiers. In this work, we used both approaches.

To analyze the classifiers, we used the accuracy metric (Acc), which measures the percentage of objects of the dataset correctly predicted by the model [Goodfellow et al. 2016]. For example, if there are 50 objects in the test dataset and a classifier correctly predicted 40 of them, the accuracy of this classifier is 80%. We used as a baseline the accuracy associated with the major class, *i.e.* the accuracy obtained when we always use the majority class to classify all given instance. For example, if we have a binary (yes or no) dataset and 80 samples in the positive class and 20 samples in the negative class, the major class is the positive one. The major accuracy in that example is

80% as the total objects are 100, and 80 of them are positive.

For sampling the dataset for evaluation, as the number of objects in our datasets was small, we used the leave-one-out cross-validation technique to separate the training and the testing objects. This technique [Japkowicz and Shah 2011] divides the dataset into N parts, each part containing only one object. It uses $N - 1$ parts for training and 1 part for testing. Thus, to evaluate a single algorithm, we have to train and validate N different models. Note that we only have one object to test for each model, so we calculate the accuracy of the whole algorithm after testing the N different models.

3. Literature Review

In a previous work [Alves and de Assumpção Drummond 2017], we proposed a multivariate and quantitative model able to predict cross-application interference level that considers the number of concurrent accesses to SLLC, DRAM and virtual network, through different virtual machines (VMs), and the similarity between the amount of those accesses. The experimental analysis showed that the model could estimate the interference, reaching an average and maximum prediction errors around 4% and 12%, and achieving errors less than 10% in approximately 96% of all tested cases. However, the user was required to give all those values precisely. The present work modifies this dataset to use categorical features and does not rely on precise information given by the user.

Ludwig *et al* [Ludwig et al. 2019] investigated the performance interference in multi-tier applications, especially web and mobile ones, where the overhead in network communication is another important issue. They used CPU, disk, memory, network, and cache metrics as variables to calculate the interference and used a four-level way to classify it.

Ren *et al* [Ren et al. 2019] and Zacarias *et al* [Zacarias et al. 2019] propose using ML techniques to predict interference in tasks and applications co-located in multi-core computers and physical datacenters. Their proposed approaches are based on data collected from performance events and hardware counters.

Meyer *et al* [Meyer et al. 2020] proposed a two-phase interference-aware classifier. The first phase uses a classification technique to determine in which of five possible classes (memory, CPU, disk, network, or cache) the object can suffer from interference. After that, there is a clustering phase, using the K-Means algorithm, to determine the level of interference that can be absent, low, moderate, or high. The authors used this combination of models in workloads that can change the behavior frequently over time.

In this work, we focus on high-performance applications, which are much more computationally intensive than web applications. So, we used three metrics to predict the interference: cache, memory, and network metrics. It uses categorical input features as it is easier for the end-user to know the overall application access' pattern in the resources than the exact number of accesses. We also rely on the uncertain data the user gives to predict the interference.

4. Interference Datasets with Uncertain Data

We created two interference datasets with uncertain data from a dataset containing real data from several co-locating applications with distinct access levels to SLLC, DRAM,

and to virtual network introduced in [Alves and de Assumpção Drummond 2017]. Firstly, we present the original dataset with its features. Then, we present the human-readable categorization and the creation of these new datasets. Finally, we present the insertion of uncertain data in the datasets.

Original dataset. The original dataset [Alves and de Assumpção Drummond 2017] is a dataset created for a regression problem and has a total of 930 objects. 432 of them represent synthetic co-located applications, and the other 498 objects represent real ones.

Concerning the synthetic application, created to represent the usual behavior of HPC applications [Alves and de Assumpção Drummond 2017], they were generated from a template that presents, alternately, two distinct and well-defined phases. The first one, called *Computation Phase*, represents the phase at which the application performs tasks involving calculation and data movement. The other one, namely *Communication Phase*, is the phase where the application exchanges information among computing pairs. From this template, an application that puts a high pressure to the shared last level of cache (SLLC), while keeping a low access level to the virtual network (NET), for example, was created. Thus, applications with distinct amounts of individual accesses were generated, considering three target access levels for each of the three shared resources.

Regarding the real applications, the original dataset considered the following ones: MUFITS [Afanasyev 2020] is employed by petroleum engineers to study the petroleum reservoir's behavior over time [Otto and Kempka 2017]; PKTM is a seismic migration method that provides a subsurface image from the earth [Melo Alves et al. 2017]; HPL (High-Performance Linpack), that solves a dense linear system of equations by applying the Lower-Upper Factorization Method with partial row pivoting; DGEMM (Double-precision General Matrix Multiply), that performs a double precision real matrix-matrix multiplication using a standard multiply method; PTRANS (Parallel Matrix Transpose), that executes a parallel matrix transpose; and FFT (Fast Fourier Transform), that computes a Discrete Fourier Transform (DFT) of one huge one-dimensional complex data vector.

Distinct metrics express the amount of individual access to each shared resource. These include the number of references to memory per second or transmitted bytes per second, and the range of those values is also different. To treat those access rates uniformly, those values were normalized in an interval between 0.0 and 1.0, where score 1.0 represents the highest possible access rates achieved by an application based on the proposed template, and score 0.0 represents no access. As the behavior of an application changes according to the number of processors (or virtual machines) allocated to it, there are a total of 57 pairs of applications and amount of processors with different amounts of access to the SLLC, DRAM, and NET. Table 1 shows some of these combinations. Note that each application were executed on a VM provided by KVM (Kernel-based Virtual Machine) and details can be found in [Alves and de Assumpção Drummond 2017].

To create the consolidated dataset where any number of co-located applications can be modeled, the accumulated score, as well as a similarity factor in each shared resource, were used. The features of this original dataset [Alves and de Assumpção Drummond 2017] are described below:

- Accumulated access to shared resources: defined as the sum of all individual ac-

Tabela 1. Normalized SLLC, DRAM and NET score for some synthetic and real applications

		Score		
Application	**# processors**	**SLLC**	**DRAM**	**NET**
S1	6	1.00	0.00	0.10
S10	6	0.30	1.00	1.00
S16	4	0.30	0.90	0.30
PTRANS.I5	6	0.18	0.21	0.32
FFT.I4	4	0.07	0.16	0.52

cess to a shared resource performed by applications co-located in the same physical machine. This accumulated access represents the total pressure put by the applications co-located in a given shared resource.

- Global similarity factor for each shared resource: is equal to the average of all similarity factors, concerning a shared resource, calculated for each pair of applications allocated to the same physical machine.
- Interference level of co-located applications: the interference level experienced by the set of applications allocated to the same physical machine is calculated as the average slowdown of applications allocated to this physical machine. The interference level is the continuous output feature of the original dataset.

In the original dataset, the 432 objects representing synthetic applications are distributed as follows: 171 objects representing two co-located applications in six processors each; 165 objects representing three co-located applications in four processors each; 84 objects representing six co-located applications in two processors and 12 objects representing 12 co-located applications in one processor each. The 498 objects representing real applications are distributed as the following: 55 objects representing two co-located applications in six processors each; 220 objects representing three co-located applications in four processors each; 210 objects representing six co-located applications in two processors and 13 objects representing 12 co-located applications in one processor each.

Human-readable categorization. In the present work, a human-readable categorization based on the Likert scale [Likert 1932] is used. The categorization is done in the individual access to each shared resource by each application. This scale was adopted because the user usually does not know the exact number of accesses to each resource.

To create the scale, the whole normalized interval (from 0 to 1) was divided into five categories of access: 'Very low' that represents the values between 0 and 0.2; Low' that represents the values between 0.2 and 0.4; 'Medium' that represents the values between 0.4 and 0.6; 'High' that represents the values between 0.6 and 0.8; and, finally, 'Very High' that represents the values between 0.8 and 1. With the 57 pairs of application and number of processors used in the original dataset (Table 1), we have at least one pair in the five categories for the SLLC access and all but the 'High' one for the DRAM and NET access. The number of pairs in each category for each shared resource is presented in Table 2.

The original dataset used only the accumulated score of each resource. Thus, the number of features for any number of co-located applications is always the same. When we categorize the individual accesses to the resources of each application, we cannot use

Tabela 2. Number of applications for each input feature category

	Score		
Category	**SLLC**	**DRAM**	**NET**
'Very low'	34	45	33
'Low'	7	1	7
'Medium'	8	6	12
'High'	3	0	0
'Very High'	6	6	6

the accumulated score. To solve it, we separated the objects of the original dataset by the number of co-located applications, generating four groups of objects.

For two co-located applications, we have 226 objects with seven features. For three co-located applications, we have 385 objects with ten features each. For six co-located applications, we have only 294 objects with 19 features each. For 12 co-located applications, we have 26 objects with 37 features each. The groups of objects representing six and 12 co-located applications cannot be datasets as they have few objects concerning the number of features. Thus, we ended up with two datasets: one for two co-located applications and one for three co-located applications.

The dataset with two co-located applications has seven features: six represent the individual access of the application to each shared resource, and the last feature is the interference level. The dataset with three co-located applications has ten features: nine represent the individual access of the application to each shared resource, and the last one is the interference level. Each access of one application to a shared resource is the expected access behavior of all virtual machines holding that application to the shared resource. The interference level is the output feature in both datasets.

To categorize the output, we first normalized the values in each dataset and used the same categories as above: 'Very low' for normalized interference of 0 to 0.2; 'Low' for values between 0.2 and 0.4; 'Medium' for normalized values of 0.4 to 0.6; 'High' for values between 0.6 and 0.8; and 'Very high' for values of 0.8 up to 1.0. The objects from the two co-located applications dataset and the ones from the three co-located applications dataset were divided into five output categories, as shown in Table 3.

Tabela 3. Number of applications in each dataset for each output category

	Datasets	
Category	**2 app.**	**3 app.**
'Very low'	120	226
'Low'	68	73
'Medium'	28	63
'High'	6	17
'Very High'	4	6
Total	226	385

As we categorize all features, two different objects can be transformed into the same one. Those objects are called duplicates or duplicate data. So, there are 50 duplicate objects in the two applications dataset, and there are 213 duplicate objects in the three applications dataset.

Uncertain data. The adopted categorization can lead to imprecision, for example, when there is some uncertainty in the application behavior and the user has to guess it. So, we inserted some noisy data in our dataset to see how the ML models deal with that possible imprecision. When the dataset has some incorrect values in the features, we can call it noisy or uncertain data. We introduced some uncertainty in our dataset to simulate possible wrong classifications by the user. To simulate the uncertain data, we randomly selected input features in some objects and changed them to the previous or next category in our 5-scale measure.

Each object is represented as a line in the dataset and each feature is represented by a column. To conduct our experiments with uncertain data, we created datasets with different numbers of features (or columns) that changed. At first, we randomly selected some objects (or lines), and then, for each of them, we chose (also randomly) some input features (or columns) to be changed. These input values were changed to the previous or the next category in our 5-scale categorization. The percentage of changed lines (objects) and columns (input features) also varied, resulting in several datasets, so that we managed to analyze our proposal in several scenarios of uncertain data. So, we created five datasets with two co-located applications and five datasets with three co-located applications. The datasets without any column changed are called datasets without noisy data or noise. The other four datasets were created with different amounts of data changed. The first one has 25% of lines, and 25% of columns changed. The second has 25% of lines, and 50% of columns changed. The third has 50% of lines and 25% of columns changed. The last one has 50% of lines and 50% of columns changed.

5. Experimental Results

We implement our tests in Python, using the *scikit-learn* package that has many ML techniques ready to be used [Pedregosa et al. 2011]. For executing the leave-one-out cross-validation, we used the *LeaveOneOut* class with the default parameters.

For executing the K-NN algorithm, we used the *KNeighborsClassifier* class with the default algorithm, uniform weights for all neighbors, and we varied the value of K from 1 to 9. For constructing the RF model, we used the *RandomForestClassifier* class with 100 trees and used the whole training dataset to create each tree. For constructing the NB model, we used the *GaussianNB* class with the default parameters. For constructing the Linear SVM, we used two classes: *SVC* with Linear kernel and *LinearSVC*. The *SVC* class uses the 'one-vs-one' approach for multi-class problems and the *LinearSVC* uses the 'one-vs-others' approach. Note that for the K-NN model, we only present the Acc value for the optimal K in each dataset.

Two co-located application datasets: The major accuracy for these datasets is 53.10% and is the baseline for all our used techniques. The Acc for each ML classification technique is presented in Table 4. The 'w. n.' column represents the dataset without any noisy data and the '25% l. 25% c.' column represents the dataset where 25% of the lines and 25% of the columns changed to noisy data. The '25% l. 50% c.' column represents the dataset where 25% of the objects had 50% of their columns changed to noisy data. The '50% l. 25% c.' and '50% l. 50% c.' columns represent the columns where 50% of the dataset's lines were changed. The first one had only 25% of the columns changed while the last one had 50% of the columns changed. Fig. 1 shows the Acc values (bars)

and the major accuracy (continuous line).

Tabela 4. Acc values for datasets with two co-located applications

	w. n.	**25% l. 25% c.**	**25% l. 50% c.**	**50% l. 25% c.**	**50% l. 50% c.**
K-NN (optimal K)	72.12%	67.70%	65.93%	64.60%	65.49%
RF	80.09%	71.68%	68.14%	66.81%	61.95%
NB	72.57%	70.80%	68.58%	70.80%	61.06%
Linear SVM 'one-vs-one'	77.43%	75.22%	68.58%	71.68%	61.50%
Linear SVM 'one-vs-others'	71.24%	65.93%	63.72%	68.58%	56.64%

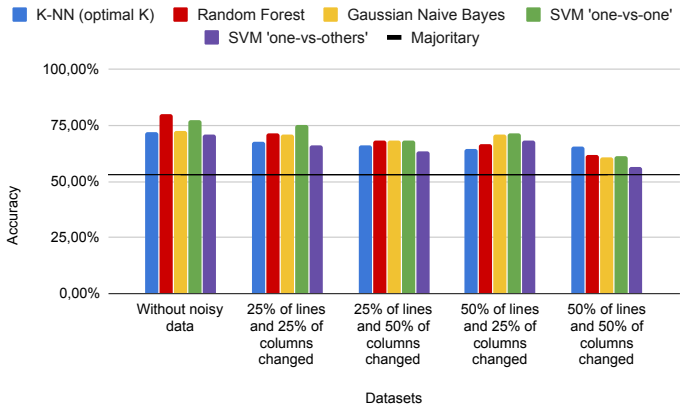

Figura 1. Learning algorithms' Acc values against major accuracy for two co-located applications dataset

Fig. 1 shows that all tested algorithms presented a larger Acc values than the major accuracy in all datasets, though, for each dataset, different learning algorithms found the highest Acc value. For the dataset without noise, the highest accuracy was in the RF, which correctly predicted almost 27% more samples than the major case, as shown in Table 4.

The Linear SVM with the 'one-vs-one' approach generated better accuracy for the datasets where 25% of the lines were modified. This accuracy was 75.22% for the one where 25% of the columns were changed. For the dataset with 50% of the columns changed, the accuracy was 68.58%. When we change 50% of the lines, different methods generated the highest Acc value. NB presented the highest accuracy of 70.80% when the noisy data is only in 25% of the columns. K-NN (with $K = 9$) presented the highest accuracy of 65.49% when we insert noise in 50% of the columns.

In the five datasets with two co-located applications, the best classifier for them is the Linear SVM using the 'one-vs-one' approach, with an average accuracy of 70.88%. The worse classifier is the Linear SVM with an average accuracy of 65.22% using the 'one-vs-others' approach. As the 'one-vs-other' approach creates only five SVMs, it can lead to unbalanced datasets, especially for the 'High' and 'Very High' classes of interference (that has only 6 and 4 objects each one). On the other hand, the 'one-vs-one'

approach creates one SVM per pair of classes, which gives the classifier a better chance to understand the output classes with few objects.

Three co-located applications datasets: The major accuracy for these datasets is 58.70% that is our baseline for the tested techniques. The accuracy for each ML classification technique is presented in Table 5. Fig. 2 shows the Acc values (bars) and the major accuracy (continuous line) for these datasets.

Tabela 5. Acc values for datasets with 3 co-located applications

	w. n.	**25% l. 25% c.**	**25% l. 50% c.**	**50% l. 25% c.**	**50% l. 50% c.**
K-NN (optimal K)	92.99%	89.09%	80.52%	85.45%	77.40%
RF	95.58%	90.91%	85.97%	86.23%	75.06%
NB	77.40%	72.47%	67.79%	73.25%	68.83%
Linear SVM 'one-vs-one'	86.75%	88.05%	79.22%	82.60%	72.73%
Linear SVM 'one-vs-others'	87.01%	86.23%	77.40%	82.08%	72.73%

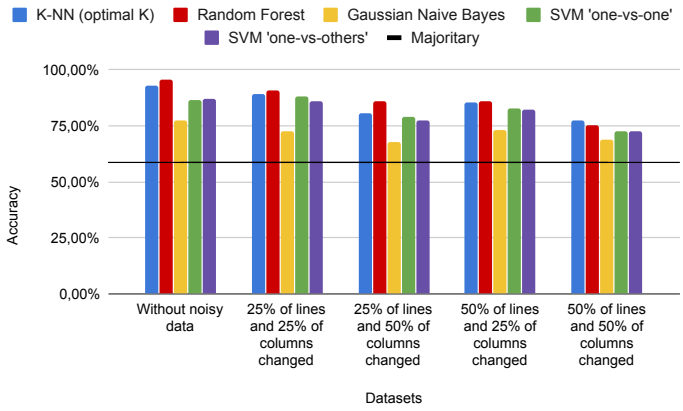

Figura 2. Learning algorithms' Acc values against major one for three co-located applications dataset

As in our previous experimenting scenario, Fig. 2 shows that the achieved Acc values of all learning algorithms were higher than the major accuracy. RF yielded the best Acc value for most of the datasets tested in this experiment. This model presented an accuracy value of 95.58% for the dataset without noise and an accuracy value of 90.91% for the first modified dataset (25% of the lines and 25% of the columns changed). It also showed an Acc value of 85.97% for the dataset where 25% of the lines and 50% of the columns were changed. RF also yielded the best Acc value of 86.23% for the dataset where 50% of the lines and 25% of its columns changed. However, K-NN (with $K = 3$) presented the best result for the last tested dataset, where 50% of its lines and 50% of the columns changed.

In the five datasets with three co-located applications, the best classifier is the RF with an average Acc value of 86.75%. The worse classifier is the NB, with an average Acc value of 71.95%. NB uses the calculated probabilities of all pairs of features input

and classes, and as our dataset is unbalanced, this probabilities can be biased. On the other hand, RF trains 100 decision trees with different random seeds and uses all of them to classify one object.

Note that as we increased the amount of noise in the dataset, Acc values did not degrade too much. It shows that these models could be used in practice when the user is not sure about the application access pattern. Also, K-NN was the best model for both datasets containing noisy data in 50% of lines and 50% of columns. It can be explained by the 50 duplicated objects in the two applications dataset and by the 213 duplicated objects presented in the three applications dataset. When we have duplicated objects, they have the same distance to all objects. Even if some noisy data were inserted in some duplicated objects, the other ones still have the same distance to the modified one, and it is what the K-NN model uses as its classification metric.

The previous work [Alves and de Assumpção Drummond 2017] that inspired this new one presented a regression model, which yielded the exact percentage of interference that the applications suffer when co-located in the same physical machine. Although the quality of the previous results is higher than ours, they used the exact number of access to predict the level of interference, and, in many cases, it is not usable in practice. The new proposed strategy can be used when the user does not have the precise numbers. This strategy presented an average Acc value of 68.35%, reaching up to 80.09% for the two applications dataset and an average Acc value of 81.35%, reaching up to 95.58% for the three applications dataset, showing that it can be an interesting alternative in those cases.

6. Conclusions

Cross-application interference is a well-known problem when co-locating applications in the same computational resource. In some scenarios, the users/providers may have a general knowledge of the execution profile of some applications, but they do not know the access rates precisely. In this work, we categorized the applications' interference in five levels and used some well-known ML techniques to predict the interference. Among them, SVM with the 'one-vs-one' and RF presented the best average accuracy of 70.88% and 86.75% for two and three co-located application datasets, respectively.

As future work, we intend to create a dataset that can predict the interference level for any number of applications. Since the performance of HPC applications executed in clouds can be improved when using a virtual placement strategy that considers the cross-application interference, we intend to implement a scheduler that uses such information to make better decisions regarding the allocation of virtual machines in data centers of cloud environments.

Acknowledgment

This research was supported by PrInt from CAPES (process 88887.310261/2018-00) and CNPq (process 145088/2019-7).

Referências

Afanasyev, A. (2013-2020). MUFITS reservoir simulation software. `http://www.mufits.imec.msu.ru/`. Last accessed in June 2020.

Alves, M. M. and de Assumpção Drummond, L. M. (2017). A multivariate and quantitative model for predicting cross-application interference in virtual environments. *Journal of Systems and Software*, 128:150 – 163.

Breiman, L. (2001). Random forests. *Machine learning*, 45(1):5–32.

Cover, T. and Hart, P. (1967). Nearest neighbor pattern classification. *IEEE transactions on information theory*, 13(1):21–27.

Cristianini, N. and Shawe-Taylor, J. (2000). *An Introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Japkowicz, N. and Shah, M. (2011). *Evaluating learning algorithms: a classification perspective*. Cambridge University Press.

Likert, R. (1932). A technique for the measurement of attitudes. *Archives of Psychology*, 22 140:55.

Ludwig, U. L., Xavier, M. G., Kirchoff, D. F., Cezar, I. B., and De Rose, C. A. F. (2019). Optimizing multi-tier application performance with interference and affinity-aware placement algorithms. *Concurrency and Computation: Practice and Experience*, 31(18):e5098. e5098 cpe.5098.

Melo Alves, M., da Cruz Pestana, R., Alves Prado da Silva, R., and Drummond, L. M. (2017). Accelerating pre-stack kirchhoff time migration by manual vectorization. *Concurrency and Computation: Practice and Experience*, 29(22):e3935.

Meyer, V., Kirchoff, D. F., da Silva, M. L., and César A. F., D. R. (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)*, pages 80–87.

Otto, C. and Kempka, T. (2017). Prediction of steam jacket dynamics and water balances in underground coal gasification. *Energies*, 10(6):739.

P. Domingos and Pazzani, M. (1997). On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. Machine Learning. *Machine Learning*, 29:103–130.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Ren, S., He, L., Li, J., Chen, Z., Jiang, P., and Li, C. T. (2019). Contention-aware prediction for performance impact of task co-running in multicore computers. *Wireless Networks*, 7.

Zacarias, F. V., Petrucci, V., Nishtala, R., Carpenter, P., and Mossé, D. (2019). Intelligent colocation of workloads for enhanced server efficiency. In *2019 31st International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 120–127.