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

Rafaela C. Brum
Fluminense Federal University
Institute of Computing
Niteroi, Brazil
rafaelabrum@id.uff.br

Flavia Bernardini
Fluminense Federal University
Institute of Computing
Niteroi, Brazil
fcbernardini@ic.uff.br

Maicon Melo Alves
PETROBRAS
Rio de Janeiro, Brazil
mmelo@ic.uff.br

Lúcia Maria de A. Drummond
Fluminense Federal University
Institute of Computing
Niteroi, Brazil
lucia@ic.uff.br

***Abstract*—Cross-application interference can drastically affect the performance of HPC applications executed in clouds. This problem is caused by concurrent access of co-located applications to shared resources such as cache and main memory. Several works of the related literature have considered general characteristics of HPC applications or the total amount of cache accesses to determine the cross-application interference. In contrast, others use metrics with numerical values and classification of the collected features, assuming that real values of the features and metrics are known. In this paper, we propose a human-readable categorization of the accesses to each shared resource based on the Likert scale. This categorization is usable in practice as the user does not know the exact number of application accesses to each resource. 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. To reflect this uncertainty, some noisy data was inserted in the used dataset that contains synthetic and real HPC co-located applications in different combinations of the number of processes and virtual machines. To predict 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). Our results showed that the tested classifiers could correctly predict the level of interference in most cases, pointing out that categorization with uncertain data can be an interesting and practical alternative to predicting application interference in cloud environments.**

***Index Terms*—Interference, Machine Learning, High-performance Computing, Virtual Machine**

I. 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 been moved from dedicated infrastructures to the computational environment offered by cloud providers more frequently.

Although HPC applications can be satisfactorily executed 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 machine, 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 [1], [2], [3], [4], [5] 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 tough information to obtain, especially for end-users. Even though using categorical input features, others did not evaluate the negative impact that uncertainty values, provided by end-users, could be produced 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, which is based on the Likert scale [6], 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

978-1-7281-4194-7/19/$31.00

and does not execute extra tests only to obtain those numbers. However, 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 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 application data with 385 objects. Our results showed that the tested classifiers can correctly predict the level of interference in up to 80.09% of the dataset with two co-located applications. When we test 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:

- An interference dataset based on a 5-scale categorization of the access to the shared resources: SLLC, DRAM, and virtual network.
- 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 II presents the main Machine Learning definitions and notations used in this work. Section III presents some related works. Section IV presents our created dataset. Section V shows the steps of our methodology for conducting our experiments. Section VI presents our experimental results. Section VII presents our conclusions and future work.

II. Machine Learning Definitions and Notations

Machine Learning (ML) is a field in the Artificial Intelligence area that tries to understand the present from previous experience [7]. Each experience collected from the real world is called an instance or object, and it is modeled as a set of features, which describes each object's main aspects. 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 $\mathbf{x}_i$. For classification purposes, y_i values values typically belong to a discrete set of L labels, *i.e.* $y_i \in \{l_1, l_2, ..., l_L\}$; for regression purposes, y_i values belong to the (sub)set of real values. Although supervised learning algorithms can be either designed for one of these purposes or both, the literature separates these kinds of problems due to the different metrics used for each kind of problem — classification or regression. 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.

In this work, 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 [8] 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 [9] trains many decision trees for each dataset, considering different random seeds. The classification of a new object is done by label voting of all trees. NB is a probabilistic model [10], 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 [11], [12]. 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 considered negative, constructing L datasets. In our case, five datasets are constructed, one for each class being considered as 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 [13]. 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 class. 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 [14] 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, as shown in Algorithm 1. 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.

Algorithm 1 Leave-one-out cross-validation

```
1: rights ← 0
2: for index ← 1 to N do
3:    train_dt ← dt without object index
4:    test_dt ← dt[index]
5:    train model with train_dt
6:    test model with test_dt
7:    if predicted category is right then
8:       rights ← rights + 1
9:    end if
10: end for
11: Acc ← rights/n
```

III. Literature Review

In a previous work [1], we showed that the cross-application interference problem is related to the amount of simultaneous access to several shared resources, revealing its multivariate and quantitative nature. Thus, in that work, 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, and the similarity between the amount of those accesses. An experimental analysis of that prediction model by using a real reservoir petroleum simulator and applications from a well-known HPC benchmark 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* [2] look to 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. They called the 'Absent' level when the interference was from 0% to 0%, 'Low' when were from 1% to 20%, 'Moderate' when were from 21% to 50% and 'High' when the interference was from 51% to 100%. They use a higher interval to the 'High' level as they have fewer examples obtaining these percentages. 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. Besides, we used a five-level scale to represent the interference and each level has a fixed interval of 20%.

Ren *et al* [3] proposed a machine learning prediction framework to understand the performance and the interference of co-located tasks in multi-core computers using 30 performance events as features. For example, two of these features were the total of branch-misses and the total number of context-switches. They developed two types of frameworks: one for tasks that are repetitively submitted to the system and another for new ones. Both regression models predicted the numerical ratio of interference when the tasks are co-located in the same physical machine. The values to the input features used in their model were taken from previous executions or from a small interval of time. Our present work 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.

Zacarias *et al* [4] proposes using ML to construct a model to predict interference in applications co-located in a physical datacenter. Their proposed ML application is based on the data collected from hardware counters and they assume that the applications have already been executed in the datacenter at least once. In our work, we do not assume any previous execution and we rely on the uncertain data the user gives us to predict the interference level of co-located applications.

Meyer *et al* [5] 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. The present work focuses on High-Performance Computing (HPC) applications that usually has a stable behavior.

IV. Interference Datasets with Uncertain Data

We created an interference dataset 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 [1]. In that original dataset, 432 objects are representing synthetic applications and 498 objects representing real ones.

Concerning the synthetic application, created to represent the usual behavior of HPC applications [15], 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 which puts a high pressure to SLLC, while keeping a low access level to the virtual network, 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. The amount of individual access to each shared resource is expressed by distinct metrics. 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 jointly, 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. These scores represent different access levels to the shared resources. Regarding the real applications, the following ones were considered in that dataset:

- MUFITS [16] is employed by petroleum engineers to study the petroleum reservoir's behavior over time [17]. From simulation results, they can make inferences about the reservoir's future conditions to maximize oil and gas production in a new or developed field. The simulator employs partial differential equations to describe the multiphase fluid flow (oil, water, and gas) inside a porous reservoir rock [18]. Reservoir simulation is one of the most expensive computational problems faced by the petroleum industry since a single simulation can take days, even weeks, to finish. The computational complexity of this problem arises from the high spatial heterogeneity of multi-scale porous media [19].
- PKTM is a seismic migration method that provides a subsurface image from the earth. This migrated image generated from a raw seismic section can reveal geological structures where oil and gas can be detected and further recovered. A raw seismic section must be migrated because the acquisition process can produce a subsurface image that originally does not represent the real geological structure found in the target area. PKTM is one of the most used migration methods in the industry due to its simplicity, efficiency, feasibility, and I/O flexibility. Similarly to the reservoir simulator, the seismic migration process is also computationally expensive and a single execution can take several hours to finish, even when executed in supercomputers [20] [21].
- HPL (High-Performance Linpack): solves a dense linear system of equations by applying the Lower-Upper Factorization Method with partial row pivoting. This application, which is usually employed to measure sustained floating point rate of HPC systems, is the basis of evaluation for the Top 500 list[1].
- DGEMM (Double-precision General Matrix Multiply): performs a double precision real matrix-matrix multiplication using a standard multiply method. Even not being a complex application, this kernel represents one of the most common operations performed in scientific computing, the matrix-matrix multiplication.
- PTRANS (Parallel Matrix Transpose): performs a parallel matrix transpose. As their pairs of processors communicate with each other simultaneously, this application is a useful test to evaluate the network's total communications capacity.
- FFT (Fast Fourier Transform): computes a Discrete Fourier Transform (DFT) of a very large one-dimensional complex data vector and is often used to measure floating point rate execution of HPC systems.

The original dataset has the following features:

- Accumulated access to shared resources: defined as the sum of all individual access to a shared resource performed by applications co-located in the same physical machine. This individual access of one application to a shared resource is defined as the sum of access of all virtual machines, holding this application to this shared resource. Thus, the accumulated access to the shared resources represents the total pressure put by the applications co-located in a given shared resource.
- Global similarity factor for each shared resource: the similarity factor of two applications regarding a given shared resource is calculated as the difference between 1 (highest individual access score) and the absolute value resultant from the difference between the number of individual accesses of two co-located applications to a shared resource. In short, the global similarity factor 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.

Since the access to each resource varies with the number of processors the application uses, the original dataset also presents access rates to each application with a different number of processors.

Thus, in the original dataset, the 432 synthetic applications objects 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 real applications

[1]https://www.top500.org/

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.

Each combination of application and number of processors gives a different access score to the shared resources. There are a total of 57 combinations, and some of them are presented in Table I. These 57 normalized access scores were the ones we based to determine the interval of each category.

TABLE I: 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

In the present work, a human-readable categorization of the access to each shared resource by each application based on the Likert scale [6] is used. This categorization was adopted because the user usually does not know the exact number of accesses to each resource. So, in this work, the whole 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. The number of access in each category for each shared resource is presented in Table II.

TABLE II: 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 original dataset used only the accumulated score of each resource. Thus, the number of features for any number of co-located applications is the same. When we categorize the individual access to the resources in each application, we cannot use an accumulated score for any applications. To solve it, we separated the objects of the original dataset by the number of co-located applications. For two co-located applications, we had 226 objects with seven features: the scores for the three shared resources for each application, and the output feature. For three co-located applications, we had 385 objects with ten features each, the scores for the three shared resources for each application, and the output feature. For six and 12 co-located applications, the number of objects was too small compared to the number of features, and thus, they were not considered ML datasets. 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 other 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 feature is the interference level. The individual access is defined as the sum of accesses of all virtual machines, holding the 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 interference of 0.4 to 0.6; 'High' for values between 0.6 and 0.8; and 'Very high' for normalized interference 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 III.

TABLE III: 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.

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.

V. Methodology

To conduct our experiments with uncertain data, we created datasets with different numbers of changed features. At first, we randomly selected some lines, and then, for each of them, we selected (also randomly) some input values 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 and columns (input values) also varied, resulting in several datasets, so that we managed to analyze our proposal in several scenarios of uncertain data. We used the value of 25% and 50% to generate the modified datasets, as follows:

- One dataset with 25% of lines and 25% of columns changed
- One dataset with 25% of lines and 50% of columns changed
- One dataset with 50% of lines and 25% of columns changed
- One dataset with 50% of lines and 50% of columns changed

In our experiments, we analyzed five datasets with two co-located applications and five datasets with three co-located applications. The dataset described in the previous section is called dataset without noisy data, or simply without noise. Our tests were implemented in Python, as there is the *scikit-learn* package implemented in this language, with many ML techniques ready to be used [22]. For executing the leave-one-out technique, 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. The 'one-vs-one' approach creates $L \times (L-1)/2$ SVMs, one for each pair of output classes and the object is classified by a voting process. The 'one-vs-others' approach creates L SVMs, one for each class as the correct one and the object are classified by the highest SVM value.

VI. Experimental Results

Firstly, we present the results for the datasets with two co-located applications and then the results for the datasets with three co-located applications. 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 IV. 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 were 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).

TABLE IV: Acc values for datasets with 2 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%

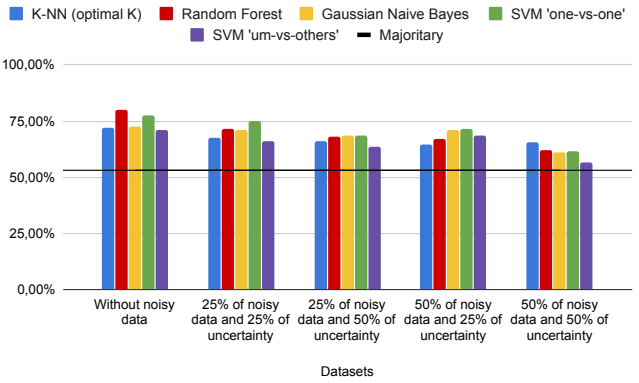

Fig. 1: Learning algorithms' Acc values against major accuracy for two co-located applications dataset

As we can see in Fig. 1, all tested algorithms presented a larger Acc values than the major accuracy in all datasets, though, for each dataset, the highest accuracy was found by different learning algorithms. 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 IV.

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 dataset 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 accuracy. 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.

When we look at 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 for them 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 for our dataset, it can lead to unbalanced datasets, especially for the 'High' and 'Very High' classes of interference (that has only 6 and 4 objects in each). 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 classes with few objects.

Three co-located applications datasets: The major accuracy for these datasets is 58.70% and is our baseline for the tested techniques. The accuracy for each ML classification technique is presented in Table V. The columns in this table have the same meaning as in Table IV. Fig. 2 shows the Acc values (bars) and the major accuracy (continuous line) for these datasets.

TABLE V: 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%

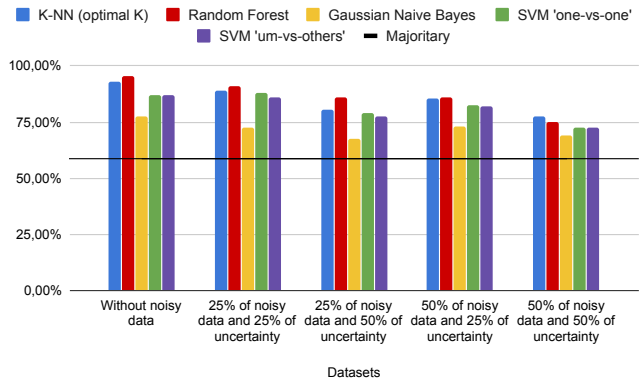

Fig. 2: Learning algorithms' Acc values against major one for three co-located applications dataset

Fig. 2 shows that, as in our previous experimenting scenario, all the achieved Acc values were greater than the major accuracy, though, for each dataset, the highest Acc value was found by different learning algorithms Besides, RF yielded the best Acc value for most of the datasets tested in this experiment. This model presented an Acc value of 95.58% for the dataset without noise, 90.91% for the datasets where 25% of the lines and 25% of the columns were changed and 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 were changed. K-NN (with $K = 3$) presented the best result for the last tested dataset, where 50% of its lines and 50% of the columns were changed.

When we look at 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 for them 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, the 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. This shows that these models could be used in practice when the user is not sure about the application access pattern. Also, the K-NN was the best model for both datasets containing noisy data in 50% of lines and 50% of columns. This 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 and it is exactly what the K-NN model uses as its classification metric.

The previous work [1] 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. That model presented an average prediction error of 4.06% and a maximum error of 12.03%. Besides, in almost 96% of all 90 tested cases, it yielded a prediction error of less than 10%. We can see that the quality of their results is higher than ours as they predict an exact number while we predict a category that represents a range of 20%. However, 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 that can be used when the user does not have the precise numbers 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.

VII. Conclusions

In this paper, we introduced interference datasets containing human-readable categories for the Shared Last Level Cache (SLLC), DRAM, and virtual network accesses. Our categorization was based on the Likert scale [6] and used 5 levels: 'Very Low', 'Low', 'Medium', 'High' and 'Very High'. We used the dataset described in [1] to insert this categorization and to create two datasets. One of them contains two co-located applications data in 226 objects and the other dataset contains three co-located applications data in 385 objects.

We also inserted some noisy data in the datasets to simulate uncertain data the user might enter. Thus, we created a total of ten datasets: five of them containing two co-located

applications data, and five with three co-located applications data, all of them with different amounts of uncertain data.

The proposed classification model categorized the applications' interference in five levels: 'Very low', 'Low', 'Medium', 'High' and 'Very High'. To predict the interference level, we used some well-known Machine Learning techniques: K-Nearest Neighbours (K-NN), RF, Naive Bayes, and Support Vector Machine (SVM). The SVM was used with two possible approaches for multi-class problems: the 'one-vs-one' approach and the 'one-vs-others' approach.

The SVM with the 'one-vs-one' approach presented the best average accuracy of 70.88% among the two co-located application datasets. In comparison, the RF presented the best average accuracy of 86.75% among the three co-located application datasets.

It is known that the cross-application interference is a problem when co-locating HPC applications, especially in clouds. 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. So, the proposed model can be used in practice by a cloud scheduler to reduce the slowdown of applications executing in virtual machines allocated to the same physical computational resource.

As future work, we intend to collect more data to use other sampling techniques for dividing the dataset in training and validating parts. Our datasets presented in this paper are small and unbalanced, as we have only four objects of the 'Very high' class and 120 in the 'Very low' one, in the two-located applications one. Observing the performance in each output class would be interesting. We also intend to create a consolidated dataset that could 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 propose 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 *Programa Institucional de Internacionalização* (PrInt) from CAPES (process number 88887.310261/2018-00) and CNPq (process number 145088/2019-7).

REFERENCES

[1] M. M. Alves and L. M. de Assumpção Drummond, "A multivariate and quantitative model for predicting cross-application interference in virtual environments," *Journal of Systems and Software*, vol. 128, pp. 150 – 163, 2017. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0164121217300699

[2] U. L. Ludwig, M. G. Xavier, D. F. Kirchoff, I. B. Cezar, and C. A. F. De Rose, "Optimizing multi-tier application performance with interference and affinity-aware placement algorithms," *Concurrency and Computation: Practice and Experience*, vol. 31, no. 18, p. e5098, 2019, e5098 cpe.5098. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.5098

[3] S. Ren, L. He, J. Li, Z. Chen, P. Jiang, and C. T. Li, "Contention-aware prediction for performance impact of task co-running in multicore computers," *Wireless Networks*, vol. 7, 2019. [Online]. Available: https://doi.org/10.1007/s11276-018-01902-7

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

[5] V. Meyer, D. F. Kirchoff, M. L. da Silva, and D. R. César A. F., "An interference-aware application classifier based on machine learning to improve scheduling in clouds," in *2020 28th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 2020, pp. 80–87.

[6] R. Likert, "A technique for the measurement of attitudes." *Archives of Psychology*, vol. 22 140, p. 55, 1932.

[7] T. M. Mitchell *et al.*, *Machine learning*. McGraw-Hill, Inc., New York, NY, USA, 1997.

[8] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE transactions on information theory*, vol. 13, no. 1, pp. 21–27, 1967.

[9] L. Breiman, "Random forests," *Machine learning*, vol. 45, no. 1, pp. 5–32, 2001.

[10] P. Domingos and M. Pazzani, "On the Optimality of the Simple Bayesian Classifier under Zero-One Loss. Machine Learning," *Machine Learning*, vol. 29, pp. 103–130. [Online]. Available: https://link.springer.com/article/10.1023/A:1007413511361

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

[12] V. Vapnik, *The nature of statistical learning theory*. Springer-Verlag New York, 2000.

[13] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[14] N. Japkowicz and M. Shah, *Evaluating learning algorithms: a classification perspective*. Cambridge University Press, 2011.

[15] J. M. Silva, C. Boeres, L. M. Drummond, and A. A. Pessoa, "Memory aware load balance strategy on a parallel branch-and-bound application," *Concurrency and Computation: Practice and Experience*, vol. 27, no. 5, pp. 1122–1144, 2015. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/cpe.3276

[16] "MUFITS reservoir simulation software," http://www.mufits.imec.msu.ru/, last accessed in June 2020.

[17] C. Otto and T. Kempka, "Prediction of steam jacket dynamics and water balances in underground coal gasification," *Energies*, vol. 10, no. 6, p. 739, 2017.

[18] D. W. Peaceman, *Fundamentals of numerical reservoir simulation*. Elsevier, 2000.

[19] B. Lu and M. F. Wheeler, "Iterative coupling reservoir simulation on high performance computers," *Petroleum Science*, vol. 6, no. 1, pp. 43–50, 2009.

[20] R. Xu, M. Hugues, H. Calandra, S. Chandrasekaran, and B. Chapman, "Accelerating kirchhoff migration on GPU using directives," in *2014 First Workshop on Accelerator Programming using Directives*. IEEE, 2014, pp. 37–46.

[21] M. Melo Alves, R. da Cruz Pestana, R. Alves Prado da Silva, and L. M. Drummond, "Accelerating pre-stack kirchhoff time migration by manual vectorization," *Concurrency and Computation: Practice and Experience*, vol. 29, no. 22, p. e3935, 2017.

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