The Journal of Systems and Software 128 (2017) 150–163

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

A multivariate and quantitative model for predicting cross-application interference in virtual environments

Maicon Melo Alves*, Lúcia Maria de Assumpção Drummond

Instituto de Computação, Universidade Federal Fluminense, Niterói, Brazil

ARTICLE INFO

Article history:
Received 23 October 2016
Revised 13 March 2017
Accepted 2 April 2017
Available online 3 April 2017

Keywords:
Cross-application interference
Virtual Machine Placement
Cloud computing
High Performance Computing

ABSTRACT

Cross-application interference can drastically affect performance of HPC applications executed in clouds. The 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 SLLC accesses to determine the cross-application interference. However, our experiments 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 this work we propose a multivariate and quantitative model able to predict cross-application interference level that considers the amount of concurrent accesses to SLLC, DRAM and virtual network, and the similarity between the amount of those accesses. An experimental analysis of our prediction model by using a real reservoir petroleum simulator and applications from a well-known HPC benchmark showed that our 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.

1. Introduction

Cloud computing has emerged as a promising alternative to execute HPC (High Performance Computing) applications. This new computational paradigm provides some attractive advantages when compared with a dedicated infrastructure, such as rapid provisioning of resources and significant reduction in operating costs related to energy, software licence and hardware obsolescence (Gholami et al., 2016; Dehury and Sahoo, 2016; Alves and Drummond, 2014; El-Gazzar et al., 2016; Kumar et al., 2016; Tsuruoka, 2016; Atif et al., 2016). In order to leverage the adoption of clouds for running HPC applications, some initiatives, such as UberCloud, have provided a free and experimental HPC on-demand service, where users can discuss their experiences on using such environment for executing their scientific applications (Gentzsch and Yenier, 2012, 2013).

However, some challenges must be overcome to bridge the gap between performance provided by a dedicated infrastructure and the one supplied by clouds. Overheads introduced by virtualization layer, hardware heterogeneity and high latency networks, for example, affect negatively the performance of HPC applications when executed in clouds (Alves and Drummond, 2014; Gupta et al., 2014; Nanos and Koziris, 2014; Chen et al., 2015; Lin et al., 2012). In addition, cloud providers usually adopt resource sharing policies that can reduce even more the performance of HPC applications. Typically, one physical server can host many virtual machines holding distinct applications (Shirvani and Ghojoghi, 2016), that may contend for shared resources like cache and main memory (Gholami et al., 2016; Gupta et al., 2013a, 2013b; Jin et al., 2015; Mury et al., 2014; Rameshan, 2016), reducing significantly their performance.

In order to address this problem, in Mury et al. (2014) a classification based on the *Thirteen Dwarfs*, which categorizes an application from its computational method, was used to decide which Dwarf classes could be co-located in a same physical machine without interference, i.e., keeping their original performances. Another work, Jin et al. (2015), classifies HPC applications according to their cache access pattern. From such classification, they claimed that a cache-pollution application, which presents weak-locality and large cache working set, should be primarily co-located with a cache-friendly application in order to alleviate cross-interference.

Those approaches, called here qualitative, because they consider general characteristics of HPC applications, do not determine precisely the cross-application interference. As further discussed in Sections 4.3 and 5, two applications belonging to the same

* Corresponding author.
E-mail addresses: mmelo@ic.uff.br (M. Melo Alves), lucia@ic.uff.br (L.M.d.A. Drummond).

http://dx.doi.org/10.1016/j.jss.2017.04.001
0164-1212/

Dwarf class or presenting the same cache access pattern can suffer distinct interference levels. Furthermore, as described in Section 4.3, a same application can present distinct interference levels in face of instances of different sizes, allowing to assert that the interference level can vary with the amount of data computed by the application.

Those findings indicate that a quantitative approach, which considers the amount of accesses to shared resources, could be a better strategy to determine more precisely the interference among applications co-located in a same physical machine. In this context, in Gupta et al. (2013b) the relation between SLLC (Shared Last Level Cache) access and resulting cross-application interference was investigated. Although that work is a step forward when compared with the others, our experimental results presented in Section 2.3 showed that other shared resources, besides SLLC, can also influence the interference level of co-located applications and should be considered as well. Moreover, all those related works only indicate whether applications should be or not co-located, but does not quantify the interference level between them.

In addition, our experimental tests described in Section 2.3 revealed that the interference can also be influenced by the similarity between the amount of applications accesses to shared resources. When the amounts of accesses to a shared resource are similar, applications evenly compete for that resource, increasing consequently the interference suffered by these applications. Thus, besides the amount of simultaneous access to shared resources, the similarity between the amounts of those accesses should also be considered when investigating the cause of interference.

In this paper, we propose a multivariate and quantitative model for predicting interference among applications co-located in a same physical machine by considering the amount of simultaneous accesses to shared resources and the similarity between the amount of applications accesses. In order to predict that interference level, our proposed model considers the effect that SLLC, DRAM (Dynamic Random Access Memory) and virtual network concurrent accesses impose in cross-application interference. Those three resources are considered particularly critical because (i) SLLC and DRAM are shared among cores of a processor and, (ii) virtual network, although not a hardware resource, is emulated by the hypervisor which is a central component shared by all virtual machines (Xu et al., 2010; Albericio et al., 2013).

In order to build this prediction model, at first we proposed an application template from which synthetic applications with distinct access patterns to each shared resource were generated. We measured the interferences, when those applications were executed concurrently, and generated an interference dataset containing these results. Then, our proposed model was built by applying the Multiple Regression Analysis (MRA), one of the most widely statistical procedures used for treating multivariate problems, over that interference dataset. Note that MRA is a powerful and well-known technique to build a model capable of predicting an unknown value of a dependent variable from the known values of multiple independent variables (Sudevalayam and Kulkarni, 2013; Atici, 2011; Hair et al., 2006; Mason and Perreault Jr, 1991; Ngo and La Puente, 2012).

We validated the prediction model by using a real petroleum reservoir simulator, called Multiphase Filtration Transport Simulator (MUFITS) (Afanasyev et al., 2016; Coco et al., 2016), and applications from a well-known computing benchmark, the High Performance Computing Challenge (HPCC) (Dongarra and Luszczek, 2004; Lin et al., 2012; Jackson et al., 2010). Those applications were executed with different instances as input and results showed that our model predicted cross-application interference with a maximum and average errors of 12.03% and 4.06%, respectively. Besides that, the prediction error was less than 10% in 95.56% for all tested cases.[1] In practice, this proposed model can be used by a Virtual Machine Placement (VMP) strategy to minimize the overall cross-application interference present in the cloud environment. More specifically, from the predicted interference level, a VMP strategy could determine which physical machines should host the incoming virtual machines so that the interference experienced by co-located applications was avoided or, at least, minimized. Thus, a VMP strategy, aware of the cross-application interference, could improve the performance of HPC applications running in cloud, especially in view of the potential increasing in the number of applications hosted in a same physical machine.[2]

Thus, the main contributions of this work are the following:

- An empirical evaluation presenting the multivariate and quantitative nature of the cross-application interference problem.
- A multivariate and quantitative model able to predict cross-application interference level that considers the amount of concurrent accesses to SLLC, DRAM and virtual network, and the similarity between the amount of applications accesses.
- An experimental analysis of our prediction model by using a real reservoir petroleum simulator and applications from a well-known HPC benchmark, run with different instances of the problems, showing that our model was able to predict accurately interference in virtual environments.

The remainder of this paper is organized as follows. Section 2 presents the experiments executed to generate an interference dataset. Section 3 describes the model for predicting interference. Experimental tests accomplished to evaluate the quality of our proposed model are presented in Section 4. Finally, related work is described in Section 5, while conclusions and future work are presented in Section 6.

2. Generating a cross-application interference dataset

In order to create an interference dataset, several co-locating synthetic applications with distinct access levels were executed and evaluated.

Those synthetic applications with distinct access patterns to each shared resource were obtained from a template with different input parameters.

Next, some preliminary concepts are presented in Section 2.1. Then, the proposed application template is described in Section 2.2. In Section 2.2.1 we present the used synthetic applications and in Section 2.3 results of concurrent executions of these synthetic applications are presented.

2.1. Preliminary concepts

We present here two fundamental definitions about cross-application interference used along this paper: the *accumulated access* to shared resources and the *interference level* suffered by co-located applications.

As defined in Eq. (1), the *individual access* (A) of an application i to a shared resource s is equal to the sum of access of all virtual machines holding this application to this shared resource, where nV_i is the total number of virtual machines hosting i and $V_{i,j,s}$ is the amount of access of virtual machine V_j to a shared resource s. In this work the amount of access to SLLC and DRAM are measured

[1] The source code of the synthetic application template as well as the input files needed for running the HPCC benchmark and the MUFITS simulator are available at http://www.ic.uff.br/~lucia/interference.tgz.

[2] In March of 2016, Intel® launched processor E5-2600 v4, codename Broadwell-EP, which is endowed with 22 cores. Thus, a physical server that supports two of these processors, such as Supermicro® 1028U-TN10RT+, provides 44 cores in total.

in terms of millions of references per second (MR/s), while the access to virtual network is expressed as the amount of megabytes transmitted per second (MB/s).

$$A_{i,s} = \sum_{j=1}^{nV_i} V_{i,j,s} \quad (1)$$

From Eq. (1), we defined the *accumulated access* (T) of all applications to a shared resource. Thus, as defined in Eq. (2), this accumulated access to a shared resource s is equal to the sum of access of all applications co-located in the same physical machine, where nA is the total number of applications co-located in a same physical machine and $A_{i,s}$ is the amount of individual access of application i to shared resource s. This accumulated access represents the pressure which all co-located applications put on shared resource in a given time interval.

$$T_s = \sum_{i=1}^{nA} A_{i,s} \quad (2)$$

We argue that the amount of accumulated access to specific shared resources is directly related to the slowdown suffered by co-located applications. In this work, slowdown is defined as the ratio of the execution time achieved by application when executed concurrently with co-located applications ($C_{i,X}$) to the one achieved from isolated execution (H_i) minus 1, where X is the set of applications co-located in a same physical machine, as shown in Eq. (3).

$$S_{i,X} = \frac{C_{i,X}}{H_i} - 1 \quad (3)$$

Then, the *cross-application interference level* is defined as the average slowdown of applications when co-located with other ones in a same physical machine.

For example, suppose that the execution times of two applications, namely A and B, in a dedicated processor were equal to 60 and 80 seconds. Suppose also that both these applications, when concurrently executed in that processor, spent 100 s. In such scenario, the slowdown of applications A and B would be, respectively, 67% and 25%, which represents how much additional time these applications needed to complete their executions when co-located in a same processor. The cross-application interference level between applications A and B would be 46%. Thus, applications A and B would suffer, in average, 46% of mutual interference when co-located in the same processor.

2.2. Generating synthetic applications

Access contention to shared resources is the main cause of interference suffered by applications co-located in the same physical machine. To study cross-application interference, the vast majority of previous papers employs real applications provided by traditional HPC benchmarks. However, a real application does not allow to control the number of accesses to each shared resource, and consequently, to evaluate systematically the relation between concurrent accesses and the resulting interference. Thus, we propose an *application template* from which synthetic applications with distinct access levels are created. From this template, we can create an application which puts a high pressure to SLLC, while keeping a low access level to virtual network, for example. Thus, a set of synthetic applications created from that template allows to observe interference in face of different levels of accesses to SLLC, DRAM and virtual network.

In order to represent the usual behavior of HPC applications (Silva et al., 2015), the proposed synthetic application template 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.

Table 1
STREAM kernels.

Name	Operation	Bytes per iteration
COPY	`a[i] = b[i]`	16
SCALE	`a[i] = b[i]*q`	16
SUM	`a[i] = b[i] + c[i]`	24
TRIAD	`a[i] = b[i] + c[i]*q`	24

The proposed synthetic application template is shown in Algorithm 1. Firstly, the synthetic application executes the *Main Loop* (lines 1 to 13) whose total number of iterations is controlled by parameter ω. This parameter leverages the total execution time of the application. Next, the synthetic application executes *Computation Phase Loop* (lines 2 to 9) at which it performs the *Computation Phase* with the number of iterations defined by α. This *Computation Phase* is based on the benchmark STREAM which is widely used to measure the performance of memory subsystems (Papagiannis and Nikolopoulos, 2014; Xavier et al., 2013). In order to measure sustainable memory bandwidth, this benchmark executes four simple vector kernels, as presented in Table 1. Because SUM presents the highest tax of memory access, it was chosen to be included in the proposed template.

The SUM operation is executed by the inner loop *Memory Access Loop* (lines 3 to 8) which is controlled by two input parameters, γ and δ. The first one defines the sizes of vectors *A, B* and *C*, and is indirectly used to determine application's Working Set Size (WSS) (Lelli et al., 2012; Gupta et al., 2013b). A small WSS usually increases application's cache hit ratio because all data needed by it in a given time interval can be entirely loaded in cache. On the other hand, when the application has a WSS greater than cache capacity, its cache hit ratio decreases because all data is fetched from main memory. Thus, WSS can be used to control the cache hit ratio and, consequently, control the number of DRAM references per second also.

The second parameter of *Memory Access Loop*, δ, controls the step at which the vector elements are accessed. Thus, when δ is equal to 1, all elements of vectors *A, B* and *C* are accessed consecutively, resulting in a high cache hit ratio. Otherwise, when δ is set to a high value, more data is fetched from main memory, resulting

Algorithm 1 Synthetic application template.

```
   Input parameters: ω, α, γ, δ, θ, β, λ
   /* Main Loop*/
1: for x=1 to ω do
       /* Computation Phase Loop*/
2:     for y=1 to α do
           /* Memory Access Loop*/
3:         for i=1 to γ step δ do
4:             A[i] = B[i] + C[i];
                   /*Compute-intensive Loop*/
5:             for k=1 to θ do
6:                 T = SquareRoot(T);
7:             end for
8:         end for
9:     end for
       /* Communication Phase Loop*/
10:    for z=1 to β do
11:        All-to-All-Communication(D,λ);
12:    end for
13: end for
```

Table 2
Application template input parameters.

Input parameter	Description
ω	Application's total number of iterations
α	Total number of iterations of Computation Phase
β	Total number of iterations of Communication Phase
γ	Working Set Size (WSS)
δ	Step at which the vector elements in *Memory Access Loop* are accessed
θ	Total number of iterations of *Compute-intensive Loop*
λ	Amount of data exchanged among processes

Table 3
Configuration of the machine used in experiments.

Model	Itautec MX214
CPU	2x Intel Xeon X5675 3.07 GHz
DRAM	48 GB DDR3 1333 MHz
Disk	5.8 TB SATA 3 GB/s
QPI	6,4 GT/s
S.O.	Ubuntu 15.04
Kernel	3.19.0-15
Hypervisor	KVM
Hardware Emulation	Qemu 2.2.0

in performance degradation. In other words, this parameter provides another way to control the cache hit ratio and to manipulate the number of DRAM references per second.

Thus, the number of DRAM references per second and application's cache hit ratio can be controlled by performing a fine tuning of both parameters γ and δ. When the application presents a high cache hit ratio, DRAM receives few references per second because data is already available in SLLC. Likewise, the number of memory references increases when application presents a low cache hit ratio.

Besides controlling DRAM access, these parameters allows to handle the number of SLLC references per second as well. When the application presents a low cache hit ratio, the number of SLLC references per second decreases. This happens because, before accessing new data, the previous referenced data, not found in SLLC, has to be fetched from DRAM. Consequently, the application's data access rate is reduced, decreasing also the number of SLLC references per second. On the other hand, when the application presents a high cache hit ratio, the number of SLLC references per second increases. Because most data is rapidly fetched from SLLC, application increases the number of memory access requests per second.

After performing the SUM operation (line 4), the synthetic application executes *Compute-intensive Loop* which repeatedly calculates the square root of variable T (line 6). This loop, whose number of iterations is defined by θ, makes the application more or less compute-intensive. Note that when θ is set to a high value, the number of memory references decreases. This happens because variable T, being frequently referenced, is kept stored in the first cache level (cache L1), preventing memory subsystem lower levels of being accessed. As a result, SLLC and DRAM references per second decreases. Thus, together with γ and δ, θ is also used to manipulate the number of DRAM and SLLC references per second.

After *Computing Phase Loop* execution, synthetic application performs *Communication Phase* by executing *Communication Phase Loop* (lines 10 to 12) whose number of iterations is determined by input parameter β. This phase is based on MP_Bench benchmark (Filgueira et al., 2012). From all MPI (Message Passing Interface) operations executed by this benchmark, `MPI_Alltoall` was particularly interesting for our work because it is widely used in scientific applications. When using this collective operation, all application's processes send and receive to/from each other the same amount of data (Steffenel et al., 2007; Hochstein et al., 2008; Li et al., 2011).

For each *Comunnication Phase Loop* iteration, the application executes *All-to-All-Communication()* function (line 11) that employs `MPI_Alltoall` of a vector D whose size is defined by the input parameter λ. So, this input parameter is used to handle the number of bytes transmitted in the virtual network.

Synthetic applications with distinct access profiles can be generated by varying properly all of these aforementioned input parameters, whose descriptions are summarized in Table 2.

Unlike adopting real applications, with this proposed synthetic application template, several applications that put distinct pressure levels to SLLC, DRAM and virtual network can be generated, providing a proper way to investigate systematically the relation between number of accesses and cross-application interference.

2.2.1. Used synthetic applications

In this section, we present the set of synthetic applications generated from the previously presented application template. 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 accesses to each shared resource is expressed by distinct metrics, such as number of references to memory per second or transmitted bytes per second, and the range of those values are also different. To treat those access rates jointly, we normalized those values 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, in our work, represent different access levels to the shared resources. Then, we created applications with *high, medium* and *low* access levels to SLLC, DRAM and virtual network, where the high access level corresponds, in our proposal, to the highest access rate to each shared resource, and medium and low access levels correspond to 50% and 10% of this high access rate, respectively.

To generate those applications with these distinct access levels, we varied input parameters of application template and monitored resulted access rates to each shared resource by using the following monitoring tools: PAPI (Performance Application Programming Interface) (Chiang et al., 2016), OProfile (Tsao and Chen, 2012) and SAR (System Activity Report) (Popiolek and Mendizabal, 2013).[3]

We executed this set of synthetic applications in a Itautec MX214 server whose configuration details are described in Table 3. As illustrated in Fig. 1, this server is equipped with two NUMA (Non-Uniform Memory Access) nodes interconnected by a QPI (Quick Path Interconnect) of 6.4 GT/s, each NUMA node has 24 GB of DRAM memory and is endowed with a Intel Xeon X5675 3.07 GHz processor. A processor has six cores that share a 12 MB SLLC unit. Moreover, the virtual environment was provided by KVM (Kernel-based Virtual Machine) hypervisor running on top of Ubuntu Server.

In our proposal, each application uses six virtual machines, each one deployed in one core, allocating 4 GB of main memory. We used CPU affinity to deploy half of the virtual machines of the application in each NUMA node, i.e., three virtual machines were deployed in "NUMA Node #1", while the others were deployed in "NUMA Node #2", in a dedicated machine. In this scenario, the access of a synthetic application to shared resources, specifically SLLC and DRAM, is balanced over two NUMA nodes, avoiding self-interference. Moreover, that configuration will be helpful to evaluate cross-application interference as discussed in the next section.

[3] Remark that, besides Oprofile and PAPI, other monitoring tools such as Perf (Tierney et al., 2012) can also be used to profile applications by collecting the information available in hardware performance counters.

Table 4
Generated synthetic applications and the corresponding execution profiles when executed in a dedicated machine.

Application	Absolute value			Score			Synthetic application template parameters						
	SLLC	DRAM	NET	SLLC	DRAM	NET	ω	α	β	γ	δ	θ	λ
S1	1635	4	300	1.0	0.0	0.1	25	120000	5200	7000	512	0	22600
S2	851	61	324	0.5	0.1	0.1	25	90000	5200	9000	1024	6	22600
S3	239	41	312	0.1	0.1	0.1	25	40000	5200	11500	2048	22	22600
S4	444	444	318	0.3	1.0	0.1	25	7500	5200	30000	512	0	22600
S5	224	224	324	0.1	0.5	0.1	25	2700	5200	39000	512	21	22600
S6	797	240	318	0.5	0.5	0.1	25	20000	5200	11800	256	2	22600
S7	1597	18	2892	1.0	0.0	1.0	25	120000	1500	7000	512	0	749568
S8	890	43	2810	0.5	0.1	1.0	25	90000	1500	9000	1024	6	749568
S9	220	49	2910	0.1	0.1	1.0	25	40000	1500	11500	2048	22	749568
S10	438	438	2832	0.3	1.0	1.0	25	7500	1500	30000	512	0	749568
S11	214	214	2892	0.1	0.5	1.0	25	2700	1500	39000	512	21	749568
S12	817	241	2838	0.5	0.5	1.0	25	20000	1500	11800	256	2	749568
S13	1575	22	1392	1.0	0.0	0.5	25	120000	150000	7000	512	0	150000
S14	890	52	1362	0.5	0.1	0.5	25	90000	150000	9000	1024	6	150000
S15	228	49	1335	0.1	0.1	0.5	25	40000	150000	11500	2048	22	150000
S16	438	438	1375	0.3	1.0	0.5	25	7500	150000	30000	512	0	150000
S17	221	221	1404	0.1	0.5	0.5	25	2700	150000	39000	512	21	150000
S18	824	239	1380	0.5	0.5	0.5	25	20000	150000	11800	256	2	150000

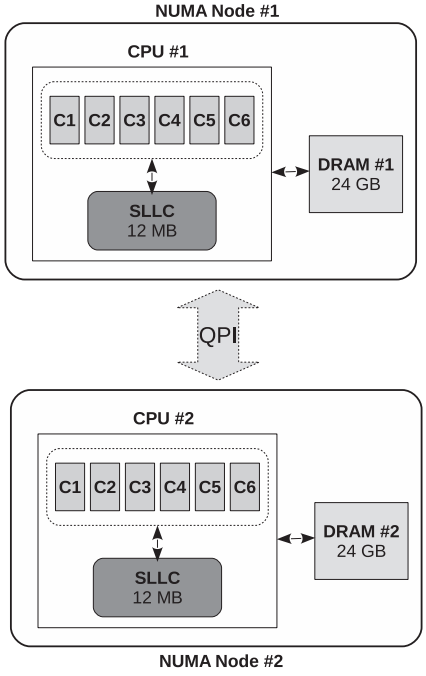

Fig. 1. NUMA nodes and processors used in our experiments.

The set of the generated synthetic applications, the corresponding execution profiles and the template input parameters, chosen to generate each application, are described in Table 4. All applications execute the same number of iterations ($\omega = 25$) and parameters α and β were set to ensure that they spent approximately the same amount of time executing *Computation* and *Communication Phases*. Moreover, all scores were rounded to one decimal place. This explains, for example, why applications S1 and S7, although having presented distinct absolute values, were classified in the same SLLC score.

Applications S1, S7 and S13 achieved the highest SLLC number of references per second. In order to reach this high SLLC individual access, we adjusted the input parameters to ensure that all memory references were directly satisfied by SLLC, which resulted in a 0.0 DRAM score. Thus, although score 0.0 was not considered as one of the three target access levels, there is no way to achieve the highest number of SLLC references per second without reducing drastically the number of accesses to DRAM.

On the other hand, the number of memory references satisfied by SLLC has to decrease to rise the number of DRAM references per second. To achieve a high DRAM individual access, all memory references should result in accesses to main memory, i.e., the SLLC hit ratio must be close to 0%. However, even in that case, the number of SLLC references per second is not equal to zero, because all references to DRAM are also treated by SLLC. This explains why applications S4, S10 and S16, which achieved the highest number of DRAM references per second, exhibited a SLLC score equal to 0.3.

Thus, concerning the memory subsystem, we were not able to generate all possible combinations involving the three access levels. A high number of individual accesses to SLLC implies in a low number of accesses to DRAM. As a consequence, it is not possible to generate an application which both SLLC and DRAM scores are equal to 1.0 or an application which puts, simultaneously, a high and medium pressure on SLLC and DRAM, for example.

Besides that, that behavior also resulted in some unexpected combinations as the one presented by application S4, that presented DRAM and SLLC scores equal to 1.0 and 0.3, respectively, though this last score was not considered as one of the target access levels.

At last, concerning virtual network, the highest amount of transmitted bytes was achieved by increasing input parameter λ up to reaching the maximum amount of data that the hypervisor is able to handle at the same time. We varied λ to find out the virtual network saturation threshold. When this limit is exceeded, the amount of bytes transmitted per second decreases, regardless of increasing λ.

Although we did not generate all possible combinations of applications, we were able to create a synthetic workload with distinct computational overhead. As can be seen in Fig. 2, this set of synthetic applications comprises a wide range of access profiles, varying from low access applications, such as S3 and S5, to the ones that put, simultaneously, a high pressure on two shared resources such as S7 and S10. We claim that this broad range of access profiles is suitable for conducting a detailed evaluation of the cross-application interference problem.

2.3. Measuring cross-applications interference

In this section, we present experiments to determine cross-applications interference. The previously presented synthetic appli-

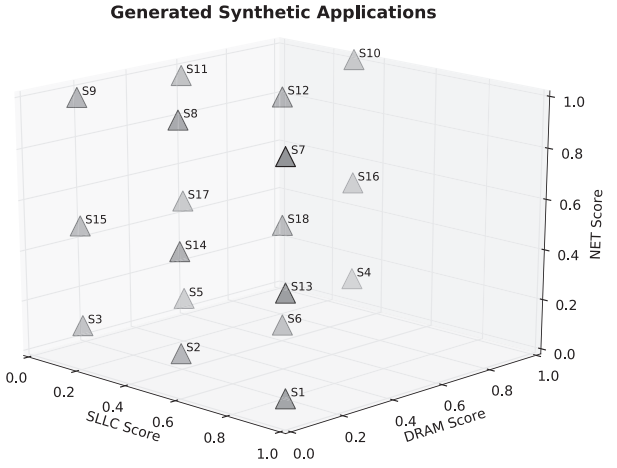

Fig. 2. Synthetic applications with distinct access profiles.

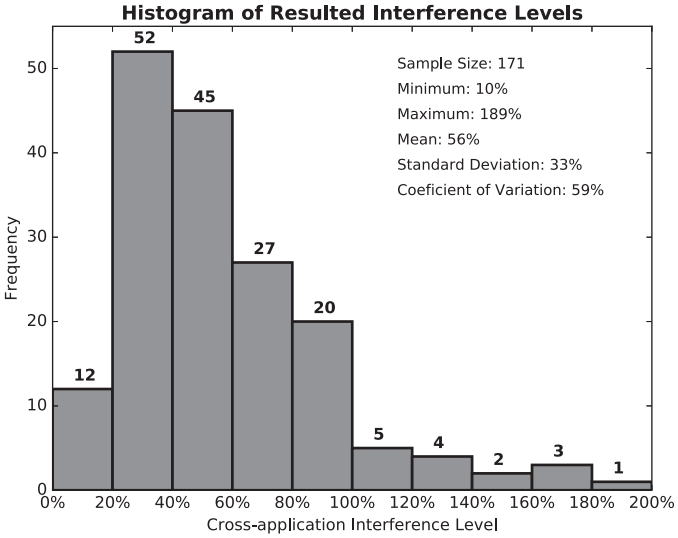

Fig. 3. Frequency of cross-applications interference levels.

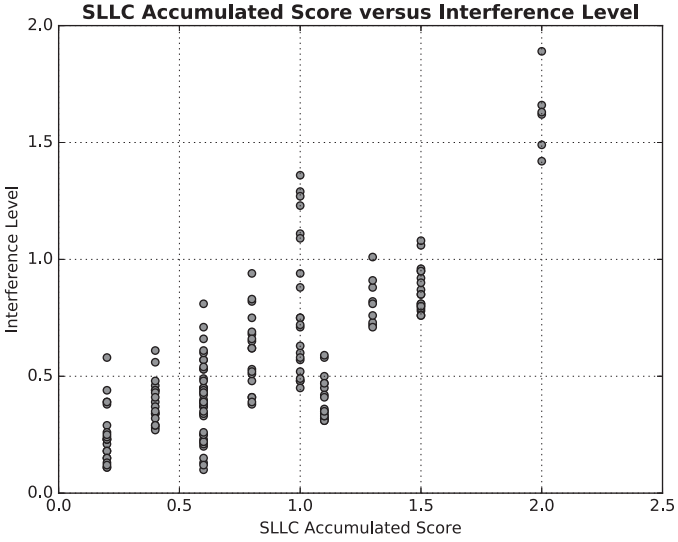

Fig. 4. SLLC accumulated score versus interference level.

cations were executed in a two-by-two fashion to obtain the resulting interference level in several cases. Because each synthetic application used half of available resources (memory and CPU), we were able to co-locate two of those applications in the physical machine, without exceeding the available resources in the system. That allocation represents a realistic scenario, usually found in clouds environments, where all resources available in a physical machine are fully allocated to maximize resource utilization (Pires and Barán, 2015; Shirvani and Ghojoghi, 2016).

The generated synthetic applications do not present exactly the same execution time, so to keep concurrency among co-located applications until the end of the experiment, the smaller execution time application was re-started automatically as many times as necessary to cover the entire execution of the longer application. We adopted such approach to fairly measure the interference suffered by both applications, regardless their execution times.

As the synthetic workload is composed of 18 applications, our interference experiments comprised 171 concurrent executions in total. From these experiments, we could observe several interesting aspects about cross-application interference. For example, the application S7, when co-located with itself, achieved the highest interference level, around 189%. This is expected because this application presents a high SLLC and virtual network access rates. On the other hand, the application S10, even presenting a high DRAM and virtual network access rates, achieved, when co-located with itself, an interference level 108% smaller than the one suffered by S7. Despite both applications present a high virtual network access rates, the application S7, which put a high pressure on SLLC, achieved a higher interference level than the one suffered by S10. These results point out that SLLC access contention imposes a higher level of interference than the one imposed by concurrent access to DRAM.

The overall interference results are summarized in Fig. 3. Around 54% of the total concurrent executions (93 occurrences) achieved an interference level less than 50%, while 37% of co-locations (63 occurrences) suffered interference levels between 50% and 100%. Besides that, in around 9% of all cases (15 occurrences) co-location applications reached interference levels greater than 100%. These results presented a coefficient of variation[4] close to 59% which allows to assert that these synthetic experiments comprised a large range of interference levels.

A deeper analysis accomplished in these results revealed that there is a correlation between interference level and SLLC accumulated score. As can be seen in Fig. 4, interference level tends to increase as SLLC accumulated score rises. Indeed, the Pearson's correlation coefficient between SLLC accumulated score and interference level is around 0.76, indicating a strong, positive and linear relationship between these both variables. Thus, this observation corroborates the hypothesis that the amount of accesses to shared resources can really influence cross-application interference.

In addition, these experiments allowed to confirm that mutual access to other shared resources besides SLLC can also impact the interference level. Consider, for example, SLLC accumulated score equal to 0.40, in Fig. 4, it may occur in cases with distinct interference levels. In other words, although SLLC accumulated access presents a strong correlation with interference level, there is another factor influencing it.

Actually, the interference level increases as virtual network also does. As illustrated in Fig. 5, when virtual network accumulated score is equal to 0.2 and 2.0, the corresponding interference levels are around 28% and 60%, respectively. For the same SLLC accumulated score, the cross-application interference levels vary more than 30% depending on the amount of access to virtual network.

Moreover, some co-locations, even though they presented almost the same amount of accumulated access to all shared re-

[4] Also known as relative standard deviation, the coefficient of variation is defined as the ratio of the standard deviation to the mean.

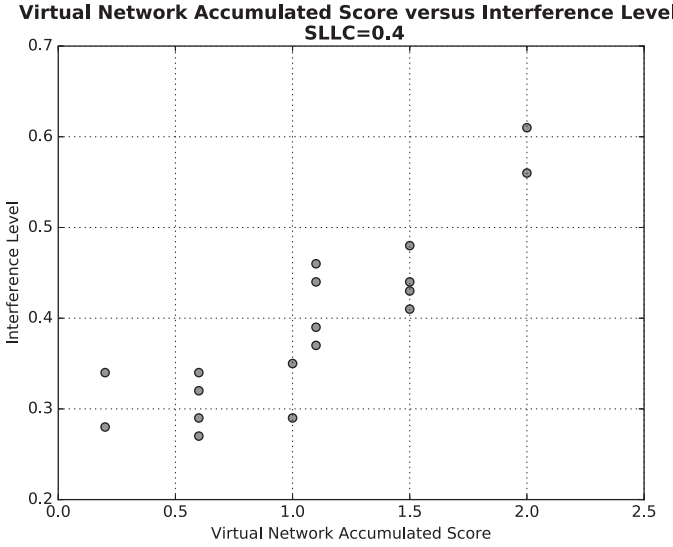

Fig. 5. Virtual network accumulated score versus interference level when SLLC was equal to 0.40.

Table 5
Results of accumulated score and interference levels for a subset of experiments.

Co-execution	Accumulated score			Interference level
	SLLC	DRAM	NET	
S1 × S3	1.1	0.1	0.2	34.10%
S2 × S2	1.0	0.2	0.2	71.12%
S13 × S3	1.1	0.1	0.6	31.51%
S14 × S2	1.0	0.2	0.6	71.83%
S15 × S7	1.0	0.2	1.5	41.67%
S14 × S8	1.1	0.1	1.5	87.97%

sources, reached interference levels that varied in more than 45%. In the subset of interference results, listed in Table 5, for example, the co-location "S14 × S8" suffered an interference level 46% higher than the co-location "S15 × S7", although both have the same virtual network accumulated score, 1.5, and differ slightly about DRAM and SLLC accumulated scores. Applications S15 and S7 present distinct SLLC individual access values, the former presents a lower number of SLLC individual access than the latter. This explains why this co-location does not present a high interference level, even achieving a high SLLC accumulated access. On the other hand, the co-location "S14 × S8", although present a SLLC accumulated access similar to co-location "S15 × S7", achieved a high interference level because both applications, which present similar SLLC individual access, evenly compete for SLLC. As can be seen in Table 5, this also happens in co-locations that present virtual network accumulated scores equal to 0.6 and 0.2.

So, besides accumulated access to shared resources, the similarity between the amounts of application's individual access has a direct impact in interference level suffered by applications when co-located in a same machine. Indeed, this justifies the difference between interference levels achieved by co-locations listed in Table 5.

In order to measure the level of similarity between two applications, we define in Eq. (4) the *similarity factor* (F). The *similarity factor* of two applications regarding to a shared resource s is calculated as the difference between 1 and the absolute value resultant from the difference between the amount of *individual accesses* (A) of applications i and j to a shared resource s.

$$F_{i,j,s} = 1 - |A_{i,s} - A_{j,s}| \tag{4}$$

Remark that, because the individual accesses (A) of applications are normalized between 0 and 1, the similarity factor will fall in this same interval. A similarity factor close to 1 indicates a high level of similarity between two applications, while values close to 0 mean that applications present distinct access profiles concerning a shared resource. In other words, the higher the similarity factor is, the higher the level of similarity between two applications will be. For example, concerning SLLC, the similarity factors for co-locations "S14 × S8" and "S15 × S7" are equal to 0.1 and 1.0, respectively. These values point out that S15 and S7 present a higher level of similarity than the one presented by S14 and S8. Indeed, as previously discussed, the applications S14 and S8 evenly compete for SLLC, while S15 and S7 put distinct access pressure on this shared resource.

From Eq. (4), we define the *global similarity factor* (G) as follows. Let X be the set of applications co-located in a same physical host, k the number of all possible pairs combinations of co-located applications, and $F_{i,j,s}$ the similarity factor of two applications, i and j, concerning a shared resource s. So, the global similarity factor, concerning a shared resource s, is defined as being the average of all similarity factors, regarding to s, calculated for each pair of co-located applications, as shown in Eq. (5).

$$G_s = \frac{\sum_{\forall i,j \in X / i \neq j} F_{i,j,s}}{k} \tag{5}$$

Results of the above described experiments showed that the cross-application interference is influenced by the following factors:

- Amount of simultaneous accesses to SLLC;
- Amount of simultaneous accesses to virtual network;
- Similarity between the amounts of applications' individual accesses to each shared resource.

As presented above, the cross-application interference problem is influenced by three different factors, which reveals its multivariate nature. Because a multivariate statistical technique allows to examine the relationship among multiple variables, it was adopted in this work to create a model for predicting the interference level suffered by co-located applications.

3. A multivariate and quantitative cross-application interference prediction model

In this section, we describe the proposed prediction model by using Multiple Regression Analysis. In Section 3.1, we briefly introduce Multiple Regression Analysis, while in Section 3.2 we describe how this technique was employed to build our prediction model.

3.1. Multiple Regression Analysis - main concepts

Multiple Regression Analysis (MRA) is a multivariate statistical technique that allows to explain the relationship between one dependent variable and, at least, two independent variables. This technique is used to create a model, called statistical variable, able to predict the value of the dependent variable from the known values of independent variables (Ngo and La Puente, 2012).

Basically, MRA comprises four macro steps as illustrated in flowchart of Fig. 6. Firstly, in *Variables Selection* step, the most likely variables to explain the behavior of the response variable are selected. Although some statistical techniques such as matrix of correlation can provide some insights about which variables must be chosen, this process is mainly guided by the researcher's knowledge about the problem (Hair et al., 2006; Ngo and La Puente, 2012).

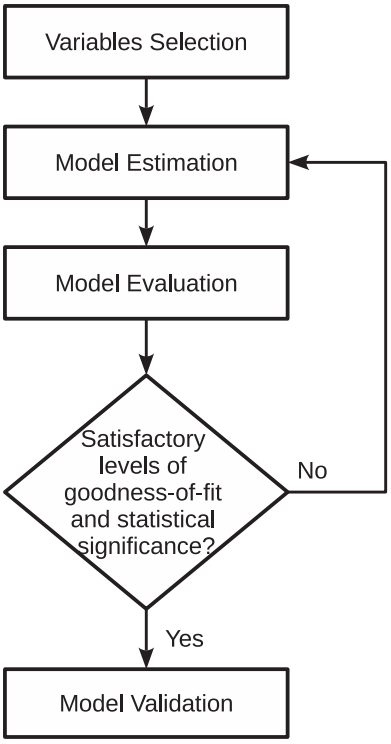

Fig. 6. Main steps of MultipleRegression Analysis.

After that, the *Model Estimation* step is executed, where the terms of the model are determined and their coefficients are automatically estimated by using the Least Squares Method. Next, the *Model Evaluation* step is executed to evaluate goodness-of-fit level, i.e., how satisfactorily the estimated model fits to data used in the building process. Besides that, statistical significance of regression and coefficients are also assessed. In case that estimated model does not present a satisfactory goodness-of-fit level or a desired statistical significance level, a new model should be estimated by executing, again, the *Model Estimation* step. Otherwise, estimated model is ready to be validated in *Model Validation* step by using an "unseen" dataset, i.e, a dataset not used previously during the process of model estimation (Hair et al., 2006; Ngo and La Puente, 2012; Mason and Perreault Jr, 1991; Vandekerckhove et al., 2014).

Although non specialized programs such as Excel and Matlab can be used for building a model from MRA, this process is usually accomplished by using commercial statistical packages such as Eviews (Min and Shengke, 2011), SAS (Agresti and Kateri, 2011), and Minitab (Wallshein and Loerch, 2015), or free software ones like Gretl (Baiocchi and Distaso, 2003) or R (R Core Team, 2016). Such specific tools provide a full multivariate analysis toolbox that allows a deep analysis on the estimated model (Wijayasiriwardhane and Lai, 2010; Nassif et al., 2013; Montgomery et al., 2015).

3.2. Building the interference prediction model

By using Minitab version 17.1.0, we followed the aforementioned steps to build our proposed model. At first, we executed the *Variables Selection* step to identify which variables should be selected as independent ones. As synthetic dataset was generated specifically to investigate interference, the selection step was straightforward. We determined accumulated scores and global similarity factors of the three shared resources as independent variables and cross-application interference level as the dependent one. We aimed at generating a model able to predict the interference level from the known values of accumulated scores and global similarity factors.

After *Variable Selection* step, we repeatedly executed both *Model Estimation* and *Model Evaluation* steps until reaching a parsimonious model with satisfactory levels of goodness-of-fit and statistical significance. A parsimonious model is able to perform better out-of-sample predictions because, due to its simplicity, it is not usually overfitted to the sample (Vandekerckhove et al., 2014).

Our multivariate and quantitative model for predicting interference is described in Eq. (6) whose terms are defined in Eqs. (7–9). As can be seen in the prediction model, the global similarity factors, G_{sllc}, G_{dram} and G_{net}, were employed to weight the influence that accumulated accesses, T_{sllc}, T_{dram} and T_{net}, impose in interference. Moreover, the total amount of access to DRAM was weighted by SLLC accumulated access since all accesses to DRAM are firstly treated in SLLC.

$$I = 0.7498 * T1 + 0.1598 * T2 + 0.1456 * T3 \tag{6}$$

where,

$$T1 = T_{sllc} * G_{sllc} \tag{7}$$

$$T2 = T_{net} * G_{net} \tag{8}$$

$$T3 = T_{dram} * G_{dram} * T_{sllc} \tag{9}$$

The coefficients calculated by each term: 0.7498 (T1), 0.1598 (T2) and 0.1456 (T3) indicate that the interference is more influenced by simultaneous and accumulated access to SLLC. However, as previously discussed, the access to virtual network and DRAM, represented by T2 and T3 respectively, that together presents a coefficient very close to 0.31, can also affect interference. Thus, depending on the value of the SLLC accumulated score, the interference can also be primarily determined by accumulated accesses to DRAM and virtual network.

That model presented an Adjusted Coefficient of Regression, that is Adjusted R-squared ($R^2 - adj$), around 0.912, which means that 91.2% of variance can be explained through that estimated model. In other words, high $R^2 - adj$ indicates that the model is suitable for the used dataset and gives accurate predictions about the dependent variable.

In addition, we assessed statistical significance of the regression model and coefficients of each term by applying the F hypothesis test. At a level of significance (α) of 0.05, test F revealed that regression and its coefficients can be considered as being statistically significant since resulted *p-values* were close to 0% (0.004). These results indicate that the probability of each coefficient has been estimated just for this sample is practically 0%. In other words, this model has almost 100% chance to be able to predict interference level for any sample besides that one used to build the model.

Moreover, an analysis accomplished on residuals showed that the estimated model did not violate any of the MRA basic assumptions. Thus, residuals presented (i) linearity, (ii) homoscedasticity and (iii) normal distribution.

4. Experimental tests and results

In this section, we describe experimental tests accomplished to assess the quality of our proposed prediction model. In Section 4.1, we describe the workload used for conducting experimental tests. In Section 4.2, we present predictions made by using our model, while in Section 4.3 we evaluate how precisely these predictions match to interference levels achieved in real experiments.

4.1. Description of the used real applications

In order to evaluate the quality of our prediction model, we devised experiments by using the following applications: a real petroleum reservoir simulator, called MUFITS, and applications from High Performance Computing Challenge Benchmark (HPCC).

MUFITS is employed by petroleum engineers to study the behavior of petroleum reservoir over time. From simulation results, they can make inferences about future conditions of the reservoir in order to maximize oil and gas production in a new or developed field. Basically, the simulator employs partial differential equations to describe the multiphase fluid flow (oil, water and gas) inside a porous reservoir rock (Peaceman, 2000; Alves and Drummond, 2014). Reservoir simulation is one of the most expensive computational problems faced by petroleum industry since a single simulation can take several days, even weeks, to finish. Computational complexity of this problem arises from the high spatial heterogeneity of multi-scale porous media (Lu and Wheeler, 2009; Yu et al., 2012; Habiballah et al., 2003).

Besides MUFITS, we tested applications from HPCC, a widely adopted benchmark to evaluate the performance of HPC systems. This benchmark provides seven kernels in total, but only four of them represent real HPC applications or operations commonly employed in scientific computing. A brief description of these four applications follows (Luszczek et al., 2006; Li et al., 2013; Younge et al., 2011; Dongarra and Luszczek, 2013).

- HPL (High Performance Linpack): solves a dense linear system of equations by applying the Lower-Upper Factorization Method with partial row pivoting. This application, that is usually employed to measure sustained floating point rate of HPC systems, is the basis of evaluation for the Top 500 list.
- DGEMM (Double-precision General Matrix Multiply): performs a double precision real matrix-matrix multiplication by using a standard multiply method. Even not being a complex real application, this kernel represents one of the most common operation 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 total communications capacity of the network.
- 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.

We considered, for each application, distinct instance sizes to certify that our proposal is able to predict interference, regardless of the input size. In MUFITS, we considered instances usually adopted in literature. The first one, labeled here as “I1”, considers a simulation of CO2 injection in the Johansen formation by using a real-scale geological model of the formation. The other one, labeled here as “I2”, is related to the 10th SPE (Society of Petroleum Engineers) Comparative Study. Both instances are available on the MUFITS Web site.[5]

For HPCC, we considered instances whose details are described in Table 6. We created these instances by adjusting parameters “#N”, “N” and “NB” which correspond, respectively, to the number of problems, the size of the problem treated by application and the size of the block. Remark that each input parameter has a specific meaning for each application. For example, in case of DGEMM, input parameter “N” is used to set the dimension of matrices to be multiplied, while this same parameter, in case of FFT, determines the size of vector of real numbers to be transformed to the frequency domain. More detailed information about these input parameters can be found on the HPCC Web site.[6]

Table 6
Description of HPCC applications instances.

Application	Instance	HPCC parameters		
		#N	N	NB
HPL	I1	1	18000	80
	I2	1	15000	80
DGEMM	I1	1	3000	80
	I2	1	18000	80
PTRANS	I1	1	24000	80
	I2	5	500	80
FFT	I1	1	65000	10
	I2	1	40000	10

4.2. Predicting interference with the proposed model

In order to assess the proposed model in distinct scenarios, we considered two co-locations schemes. In the first one, namely A, applications were co-located in a two-by-two fashion, where each application was executed with instances I1 and I2. In this scenario, each application used 6 virtual machines, except for FFT that used 4 virtual machines since this application restricts the number of processes to a power of two. In the second co-location scheme, namely B, applications were co-located in a three-by-three fashion, where each application used 4 virtual machines. Unlike A, in scenario B each application solved just instance I1. Moreover, each virtual machine used in both schemes has the same configuration as described in Section 2.2.1 and, as performed in synthetic experiments, half of virtual machines allocated to each application was deployed to a NUMA node.

To predict cross-application interference, it was necessary to obtain access rates to each shared resource when each application was executed individually. As described in Table 7, FFT achieved the highest access rate to virtual network, while PTRANS, when solving I1, imposed the highest pressure to SLLC and DRAM. Moreover, as DGEMM is a embarrassingly parallel application, it presented a low access level to virtual network. Remark that some applications decreased their access rates to shared resources when executing with four processors instead of six. This is expected since the same application, when executed with a lower number of processes, decreases, usually, the amount of individual access to shared resources.

Considering the individual profile of each application, we applied our model to predict what would be the interference suffered by those applications when co-located in accordance with schemes A and B. Prediction results point out that the minimum and maximum predicted interference levels would be equal to 1.53% and 43.10%, respectively. As expected, the lowest and highest interference levels were predicted for co-locations that involved, respectively, applications with low and high access rates to SLLC, DRAM and virtual network. In other words, our model indicated that DGEMM and HPL would suffer a low cross-interference, while FFT and PTRANS would present the highest interference levels.

However, specifically to PTRANS, our model points out that interference suffered by this application would vary significantly depending on the instance being solved. Our model indicated that PTRANS.I1.P6 would experience an interference level around 40% when co-located with itself in a two-by-two fashion, while PTRANS.I2.P6, when co-located in same conditions, would present an interference level of approximately 12%.

[5] http://www.mufits.imec.msu.ru/.

[6] http://icl.cs.utk.edu/hpcc/.

Table 7
SLLC, DRAM and virtual network individual scores of real applications executions.

Executed application	Application	Instance	Processes	Score		
				SLLC	DRAM	NET
MUFITS.I1.P6	MUFITS	I1	6	0.05	0.13	0.00
MUFITS.I2.P6	MUFITS	I2	6	0.03	0.00	0.01
MUFITS.I1.P4	MUFITS	I1	4	0.05	0.08	0.00
HPL.I1.P6	HPL	I1	6	0.03	0.06	0.02
HPL.I2.P6	HPL	I2	6	0.03	0.06	0.02
HPL.I1.P4	HPL	I1	4	0.02	0.04	0.01
DGEMM.I1.P6	DGEMM	I1	6	0.02	0.02	0.00
DGEMM.I2.P6	DGEMM	I2	6	0.01	0.02	0.00
DGEMM.I1.P4	DGEMM	I1	4	0.01	0.02	0.00
PTRANS.I1.P6	PTRANS	I1	6	0.18	0.21	0.32
PTRANS.I2.P6	PTRANS	I2	6	0.02	0.04	0.02
PTRANS.I1.P4	PTRANS	I1	4	0.14	0.09	0.19
FFT.I1.P4	FFT	I1	4	0.07	0.17	0.49
FFT.I2.P4	FFT	I2	4	0.07	0.16	0.52

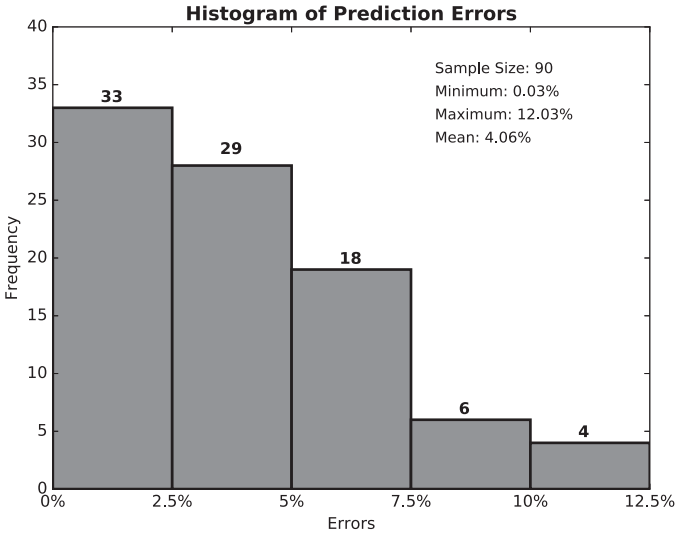

Fig. 7. Frequency of prediction errors.

Moreover, our model predicted that PTRANS and DGEMM, although belonging to the same Dwarf class, namely Dense Linear Algebra, would present distinct interference levels when co-located with themselves. So, prediction results indicated that interference level resulted from co-location "PTRANS.I1.P6xPTRANS.I1.P6" would be approximately equal to 40%, while "DGEMM.I1.P6xDGEMM.I1.P6" would suffer an interference level close to 3%.

Furthermore, our model predicted that FFT.I1.P4 and MUFITS.I1.P4, although presenting similar SLLC access rates, would suffer distinct interference levels when co-located with themselves in a three-by-three fashion. Specifically, our solution predicted that FFT.I1.P4 would present a mutual interference around 40%, while MUFITS.I1.P4 would suffer a cross-interference approximately equal to 12%.

4.3. Analysis of the interference prediction model

In order to evaluate the quality of our proposed model, we executed all of the co-locations defined in schemes A and B and, for each co-location, we calculated the *prediction error* achieved by our model. The prediction error is defined as the absolute value of the difference between interference level predicted by our model and the real interference level suffered by applications.

Table 8
Prediction errors for a subset of interference experiments with real applications.

Co-location	Interference level		Prediction error
	Real	Predicted	
PTRANS.I1.P6xPTRANS.I1.P6	44.50%	39.97%	4.53%
PTRANS.I2.P6xPTRANS.I2.P6	5.31%	12.11%	6.80%
DGEMM.I1.P6xDGEMM.I1.P6	7.79%	2.50%	5.29%
FFT.I1.P4xFFT.I1.P4xFFT.I1.P4	49.31%	40.45%	8.87%
MUFITS.I1.P4xMUFITS.I1.P4xMUFITS.I1.P4	22.85%	11.65%	11.20%

As can be seen in Fig. 7, our model presented average and maximum prediction errors equal to 4.06% and 12.03%, respectively. Moreover, in approximately 96% of all tested cases, our model presented a prediction error less than 10%. Such results revealed that our model, although being built from a two-by-two fashion experiment, could predict cross-interference for cases at which three applications were executed simultaneously, as well. It is worth mentioning that the error difference between co-locations schemes A and B was very small, the former presented average error equal to 3.39%, while the later achieved 5.10%. However, additional and exhaustive tests should be accomplished to determine precisely the impact of co-locating a varying number of applications in a same physical machine.

Some interesting results are highlighted in Table 8. At first, as predicted by our model, experimental results showed that PTRANS really presented distinct interference levels when treating I1 and I2. Our model was able to predict interference of PTRANS regardless of the instance size because it takes into account the amount of accesses of applications to shared resources. Indeed, as can be seen in Table 7, the amount of individual access of PTRANS to all shared resources when solving I1 is significantly higher than the one achieved when treating I2.

Besides that, those experimental results confirmed that PTRANS and DGEMM, even belonging to the same Dwarf class, presented distinct interference levels. This result allows to state that a qualitative approach based on Dwarfs classes is not enough to precisely determine interference, while our model was able to predict that PTRANS and DGEMM would present, respectively, a high and a low interference levels.

At last, results also showed that SLLC access contention is not the only cause for the cross-interference problem. FFT.I1.P4 and MUFITS.I1.P4, although having a similar amount of SLLC individual accesses, suffered distinct interference levels. As depicted in Table 8, our model predicted satisfactorily interference suffered by these applications because it evaluates not only SLLC, but accesses

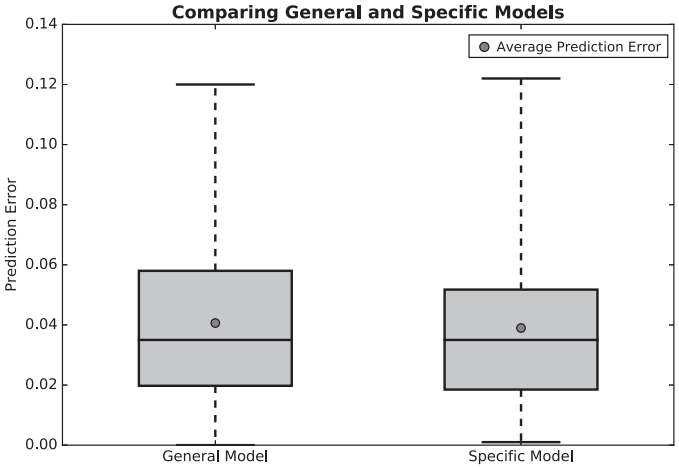

Fig. 8. Comparing prediction errors achieved by specific and general models.

to DRAM and virtual network as well. Indeed, although both applications present similar SLLC access rates, FFT.I1.P4, that suffered a higher interference, puts a higher pressure on virtual network than MUFITS.I1.P4.

All these findings allow to assert that a solution based on Dwarfs classes or that evaluates just SLLC access contention is not suitable for determining interference of co-located applications. Our proposed model satisfactorily predicted interference for these specific cases because it takes into account the amount of simultaneous access to SLLC, DRAM and virtual network, besides considering similarities between the amount of co-located application accesses.

4.4. Comparing general and specific prediction models

Results described in Section 4.3 showed that our proposed model was able to predict the interference level suffered by a set of real applications, even though those applications were not originally used in the model building process. In other words, those results point out that our proposed model is able to perform good out-of-sample predictions.

Despite those satisfactory results, we accomplished additional tests to evaluate the precision of our general model. Thus, a specific model was built by using a dataset provided by the real experiments and compared with our general model (built as previously described in Section 3.2). The validation steps for both models were executed by using the same dataset adopted in the building of the specific model.

That specific model, which is described in Eq. (10), presents the same terms of the general model and achieved an $R^2 - adj$ around 93.8%, which is slightly higher than the one achieved by the general model (91.2%).

$$I = 0.7740 * T1 + 0.1194 * T2 + 0.4190 * T3 \tag{10}$$

where,

$$T1 = T_{sllc} * G_{sllc} \tag{11}$$

$$T2 = T_{net} * G_{net} \tag{12}$$

$$T3 = T_{dram} * G_{dram} * T_{sllc} \tag{13}$$

As can be seen in Fig. 8, the specific model presented an average error lower than the one achieved by the general model, as already expected, since the specific model was built and validated by using the same dataset. However, the difference was not significant, confirming that the general model is able to accomplish very good predictions.

5. Related work

In this section, previous papers that investigated or just introduced the cross-application interference problem are presented.

Some works presented the cross-application interference, but did not propose a solution to determine or alleviate this problem. Yokoyama (2015) and Basto (2015) accomplished interference experiments by using benchmark applications in order to generate a static interference matrix. Such matrix was further used as the basis for their proposed interference-aware VMP strategies. They evaluated their VMP strategies just using applications previously used in interference experiments. Thus, unlike our work, those papers did not present a solution to determine interference among co-located applications.

Other works, besides introducing interference, proposed a naive or limited approach to explain this problem. Jersak and Ferreto (2016) devised a simple interference model to be used as a proof of concept with its proposed VMP strategy. In such model, the level of interference is defined as a function of the number of virtual machines co-executing in a physical machine. So, the model considers that the higher the number of virtual machines, the higher the interference level will be. This naive strategy is not able to determine interference since our experimental results showed that, for the same number of virtual machines, the interference can vary drastically. This happens because interference is actually related to the amount of access to shared resources and not to the number of virtual machines being co-executed in a host.

Rameshan et al. (2014) proposed a mechanism to prevent latency sensitive applications from being adversely affected by best-effort batch applications when co-located in a same physical machine. In such proposal, latency sensitive applications report to mechanism whenever they are under interference. Then, the mechanism uses information collected in that instant to predict when latency application will suffer interference again. From this prediction, the mechanism throttles the batch application before it imposes interference to latency application one more time. Thus, that work just proposed a way to work around the interference suffered by a specific class of application by monitoring the conditions that lead to occurrence of the interference. So, the root of interference problem was not investigated and, as a consequence, a solution to determine interference suffered by any set of applications was not proposed.

Other works, however, investigated the cross-application interference problem in a broader sense and proposed solutions to determine or alleviate interference experienced by co-located applications in general.

Mury et al. (2014) argued that cross-application interference could be determined by adopting a classification of applications. Such qualitative approach is based on the *Thirteen Dwarfs* which classifies applications according to computational methods usually adopted in scientific computing. Such classification is not suitable for determining interference because, as described in Section 4.3, two applications belonging to Dense Linear Algebra class can present distinct interference levels. Besides that, those results also pointed out that a same application, namely PTRANS, can present distinct interference levels when solving instances with different sizes. Actually, experiments conducted in Gupta et al. (2013b) also showed this same behavior when applications EP (Embarrassingly Parallel) and LU (Lower-Upper Gauss–Seidel solver)[7] presented dis-

[7] Applications EP, LU, CG, IS and FT are provided by the Nas Parallel Benchmark (NPB).

tinct interference levels when solving also instances of different sizes.

Jin et al. (2015) classified applications in three SLLC access classes called (i) cache-pollution, (ii) cache-sensitive and (iii) cache-friendly. These classes were further used to propose a VMP strategy with goal to alleviate interference by co-located applications with compatible SLLC access profiles. That work claimed that cache-pollution applications should be preferably co-located with cache-friendly applications rather than being co-located with cache-sensitive ones. However, through some practical experiments they showed that approach may fail. More specifically, EP and IS (Integer Sort), classified as cache-friendly, suffered distinct interference levels when co-located with CG (Conjugate Gradient), categorized as a cache-pollution application. Besides that, although both CG and FT (fast Fourier Transform) were classified as cache-pollution applications, CG did not present interference when co-located with itself, while FT suffered an interference when co-located with CG. These results show that a qualitative approach based on SLLC access pattern is not suitable for determining cross-application interference precisely.

Unlike the aforementioned works, Gupta et al. (2013b) adopted a quantitative approach to investigate cross-application interference. They studied the relation between interference and the number of simultaneous accesses to SLLC. As a result, they identified a maximum access limit that ensures a free interference co-location. Although this work proposed a quantitative strategy to explain interference problem, only one shared resource, SLLC, was considered. As previously presented in Section 2.3, our experiments showed that other shared resources such as virtual network can also influence interference and, consequently, should be systematically evaluated. Thus, a solution that considers only the concurrent access to SLLC is not suitable for determining precisely the cross-interference. However, it is worth mentioning that our proposal was build up on that work. We employed their quantitative approach to investigate the relation between the level of interference suffered by co-located applications and the amount of simultaneous access to other shared resources besides SLLC.

Moreover, two of the previously described works, by Gupta et al. (2013b) and by Jin et al. (2015), just inform whether applications should be or not co-located, without quantifying the level of interference suffered by them. However, the information about interference level can enrich the VMP process, specially in cases where only applications presenting high cross-interference levels are available.

To the best of our knowledge, this paper is the first one that proposes a multivariate and quantitative model for predicting cross-application interference level by considering accesses to SLLC, DRAM and virtual network, jointly. We claim that a quantitative approach, that considers the amount of accesses of co-located applications to shared resources simultaneously, is more suitable for determining cross-application interferences.

6. Conclusions and future work

In this paper, we presented a multivariate and quantitative model that takes into account the amount of simultaneous access to SLLC, DRAM and virtual network, and the similarity of the amount of applications individual accesses to predict the cross-applications interferences. Experimental results, considering a real petroleum reservoir simulator and applications from HPCC benchmark, showed that the proposed model was able to predict interference with an average and maximum errors around 4% and 12%, respectively. Besides that, in 96% of all tested cases, our model reached a prediction error less than 10%.

More specifically, our experimental tests showed that the model could correctly predict cross-applications interferences in different scenarios. It was able to precisely predict interference regardless of the number of applications being co-executed in the same machine and their instance sizes. Moreover, our model could predict interference for co-locations that, even presenting similar amount of accumulated accesses to SLLC, achieved distinct interference levels. Our model accurately predicted interference in that case because it considers, besides SLLC, accesses to DRAM and virtual network.

From those experimental results, we claim that HPC applications can be satisfactorily executed in clouds because the negative impact imposed by interference can be avoided or minimized by co-locating applications with complementary access profiles. So, a VMP strategy, aware of the cross-application interference problem, would be able to prevent applications from being adversely affected by other ones co-located in a same physical machine.

In future work, we expect to evaluate the influence that concurrent access to other shared resources imposes in cross-application interference. More specifically, we plan to consider in our prediction model the amount of access to disk since the concurrent access to this shared resource has a devastating influence in cross-application interference (Dorier et al., 2014).

In addition, we are also interested in investigating whether the total amount of allocated memory has a direct impact in cross-application interference. We are also interested in assessing other metrics of virtual network such as the number of packets transmitted, for example.

Besides that, we intend to evaluate whether interference varies in distinct hardware configurations and with a varying number of co-located applications to possibly incorporate those issues into our prediction model by using, for example, online machine learning techniques (Fontenla-Romero et al., 2013). Thus, the model coefficients could be dynamically adjusted concerning those specific characteristics in the cloud environment.

References

Afanasyev, A., Kempka, T., Kühn, M., Melnik, O., 2016. Validation of the MUFITS reservoir simulator against standard industrial simulation tools for CO2 storage at the Ketzin Pilot site. In: EGU General Assembly Conference Abstracts, 18, p. 6883.

Agresti, A., Kateri, M., 2011. Categorical Data Analysis. Springer.

Albericio, J., Ibáñez, P., Viñals, V., Llabería, J.M., 2013. The reuse cache: downsizing the shared last-level cache. In: Proceedings of the 46th Annual International Symposium on Microarchitecture. ACM, pp. 310–321.

Alves, M., Drummond, L., 2014. Análise de desempenho de um simulador de reservatórios de petróleo em um ambiente de computação em nuvem. XV Simpósio em Sistemas Computacionais de Alto Desempenho (WSCAD 2014) - Brazil.

Atici, U., 2011. Prediction of the strength of mineral admixture concrete using multivariable regression analysis and an artificial neural network. Expert Systems with Applications 38 (8), 9609–9618.

Atif, M., Kobayashi, R., Menadue, B.J., Lin, C.Y., Sanderson, M., Williams, A., 2016. Breaking HPC barriers with the 56GbE cloud. Procedia Computer Science 93, 3–11.

Baiocchi, G., Distaso, W., 2003. GRETL: econometric software for the GNU generation. Journal of Applied Econometrics 18 (1), 105–110.

Basto, D.T., 2015. Interference Aware Scheduling for Cloud Computing. Universidade do Porto Master's thesis.

Chen, L., Patel, S., Shen, H., Zhou, Z., 2015. Profiling and understanding virtualization overhead in cloud. In: 44th International Conference on Parallel Processing (ICPP). IEEE, pp. 31–40.

Chiang, M., Yang, C., Tu, S., 2016. Kernel mechanisms with dynamic task-aware scheduling to reduce resource contention in NUMA multi-coresystems. Journal of Systems and Software 121, 72–87.

Coco, A., Gottsmann, J., Whitaker, F., Rust, A., Currenti, G., Jasim, A., Bunney, S., 2016. Numerical models for ground deformation and gravity changes during volcanic unrest: simulating the hydrothermal system dynamics of a Restless Caldera. Solid Earth 7 (2), 557.

Dehury, C.K., Sahoo, P.K., 2016. Design and implementation of a novel service management framework for IoT devices in cloud. Journal of Systems and Software 119, 149–161.

Dongarra, J., Luszczek, P., 2013. HPC challenge: design, history, and implementation highlights. Contemporary High Performance Computing: From Petascale Toward Exascale.

Dongarra, J.J., Luszczek, P., 2004. Introduction to the HPC Challenge Benchmark Suite. Technical Report. DTIC Document.

Dorier, M., Antoniu, G., Ross, R., Kimpe, D., Ibrahim, S., 2014. CALCioM: mitigating I/O interference in HPC systems through cross-application coordination. In: 28th International Parallel and Distributed Processing Symposium. IEEE, pp. 155–164.
El-Gazzar, R., Hustad, E., Olsen, D.H., 2016. Understanding cloud computing adoption issues: a Delphi study approach. Journal of Systems and Software 118, 64–84.
Filgueira, R., Carretero, J., Singh, D.E., Calderón, A., Núñez, A., 2012. Dynamic-CoMPI: dynamic optimization techniques for MPI parallel applications. The Journal of Supercomputing 59 (1), 361–391.
Fontenla-Romero, Ó., Guijarro-Berdiñas, B., Martinez-Rego, D., Pérez-Sánchez, B., Peteiro-Barral, D., 2013. Online machine learning. Efficiency and Scalability Methods for Computational Intellect, 27.
Gentzsch, W., Yenier, B., 2012. The Uber-Cloud Experiment. HPCwire, June 28.
Gentzsch, W., Yenier, B., 2013. The UberCloud HPC Experiment: Compendium of Case Studies. Technical Report. Tabor Communications.
Gholami, M.F., Daneshgar, F., Low, G., Beydoun, G., 2016. Cloud migration process - asurvey, evaluation framework, and open challenges. Journal of Systems and Software 120, 31–69.
Gupta, A., Faraboschi, P., Gioachin, F., Kale, L.V., Kaufmann, R., Lee, B., March, V., Milojicic, D., Suen, C.H., 2014. Evaluating and improving the performance and scheduling of HPC applications in cloud. IEEE Transactions on Cloud Computing 7161 (c).
Gupta, A., Kale, L.V., Gioachin, F., March, V., Suen, C.H., Faraboschi, P., Kaufmann, R., Milojicic, D., Lee, B., 2013. The Who, What, Why and How of High Performance Computing Applications in the Cloud. HP Laboratories.
Gupta, A., Kale, L.V., Milojicic, D., Faraboschi, P., Balle, S.M., 2013. HPC-aware VM placement in infrastructure clouds. In: International Conference on Cloud Engineering (IC2E). IEEE, pp. 11–20.
Habiballah, W., Hayder, M., Khan, M., Issa, K., Zahrani, S., Shaikh, R., Uwaiyedh, A., Tyraskis, T., Baddourah, M., et al., 2003. Parallel reservoir simulation utilizing PC-clusters in massive reservoir simulation models. In: Annual Technical Conference and Exhibition. Society of Petroleum Engineers (SPE).
Hair, J.F., Black, W.C., Babin, B.J., Anderson, R.E., Tatham, R.L., 2006. Multivariate Data Analysis (Vol. 6). Upper Saddle River, NJ: Pearson Prentice Hall.
Hochstein, L., Basili, V.R., Vishkin, U., Gilbert, J., 2008. A pilot study to compare programming effort for two parallel programming models. Journal of Systems and Software 81 (11), 1920–1930.
Jackson, K.R., Ramakrishnan, L., Muriki, K., Canon, S., Cholia, S., Shalf, J., Wasserman, H.J., Wright, N.J., 2010. Performance analysis of high performance computing applications on the Amazon web services cloud. In: Second International Conference on Cloud Computing Technology and Science (CloudCom). IEEE, pp. 159–168.
Jersak, L.C., Ferreto, T., 2016. Performance-aware server consolidation with adjustable interference levels. In: Proceedings of the 31st Annual Symposium on Applied Computing. ACM, pp. 420–425.
Jin, H., Qin, H., Wu, S., Guo, X., 2015. CCAP: a cache contention-aware virtual machine placement approach for HPC cloud. International Journal of Parallel Programming 43 (3), 403–420.
Kumar, A., Sathasivam, C., Periyasamy, P., 2016. Virtual machine placement in cloud computing. Indian Journal of Science and Technology 9 (29).
Lelli, J., Faggioli, D., Cucinotta, T., Lipari, G., 2012. An experimental comparison of different real-time schedulers on multicore systems. Journal of Systems and Software 85 (10), 2405–2416.
Li, Q., Huo, Z., Sun, N., 2011. Optimizing MPI Alltoall communication of large messages in multicore clusters. In: 12th International Conference on Parallel and Distributed Computing, Applications and Technologies. IEEE, pp. 257–262.
Li, Z., Zhang, H., OBrien, L., Cai, R., Flint, S., 2013. On evaluating commercial cloud services: a systematic review. Journal of Systems and Software 86 (9), 2371–2393.
Lin, Q., Qi, Z., Wu, J., Dong, Y., Guan, H., 2012. Optimizing virtual machines using hybrid virtualization. Journal of Systems and Software 85 (11), 2593–2603.
Lu, B., Wheeler, M.F., 2009. Iterative coupling reservoir simulation on high performance computers. Petroleum Science 6 (1), 43–50.
Luszczek, P.R., Bailey, D.H., Dongarra, J.J., Kepner, J., Lucas, R.F., Rabenseifner, R., Takahashi, D., 2006. The HPC challenge (HPCC) benchmark suite. In: Proceedings of the 2006 ACM/IEEE Conference on Supercomputing. Citeseer, p. 213.
Mason, C.H., Perreault Jr, W.D., 1991. Collinearity, power, and interpretation of multiple regression analysis. Journal of Marketing Research 268–280.
Min, L., Shengke, C., 2011. Eviews Statistical Analysis and Applications. Electronic industry, pp. 5–10.
Montgomery, D.C., Peck, E.A., Vining, G.G., 2015. Introduction to Linear Regression Analysis. John Wiley & Sons.
Mury, A.R., Schulze, B., Licht, F.L., de Bona, L.C., Ferro, M., 2014. A concurrency mitigation proposal for sharing environments: an Affinity approach based on applications classes. In: Intelligent Cloud Computing. Springer, pp. 26–45.
Nanos, A., Koziris, N., 2014. Xen2MX: high-performance communication in virtualized environments. Journal of Systems and Software 95, 217–230.
Nassif, A.B., Ho, D., Capretz, L.F., 2013. Towards an early software estimation using log-linear regression and a multilayer perceptron model. Journal of Systems and Software 86 (1), 144–160.
Ngo, T.H.D., La Puente, C., 2012. The steps to follow in a multiple regression analysis. In: Proceedings of the SAS Global Forum 2012 Conference (paper 333-2012). Citeseer.
Papagiannis, A., Nikolopoulos, D.S., 2014. Hybrid address spaces: a methodology for implementing scalable high-level programming models on non-coherent many–core architectures. Journal of Systems and Software 97, 47–64.
Peaceman, D.W., 2000. Fundamentals of Numerical Reservoir Simulation. Elsevier.
Pires, F.L., Barán, B., 2015. A virtual machine placement taxonomy. In: 15th International Symposium on Cluster, Cloud and Grid Computing (CCGrid). IEEE/ACM, pp. 159–168.
Popiolek, P.F., Mendizabal, O.M., 2013. Monitoring and analysis of performance impact in virtualized environments. Journal of Applied Computing Research 2 (2), 75–82.
R Core Team, 2016. R: A Language and Environment for Statistical Computing. R Foundation for Statistical Computing, Vienna, Austria.
Rameshan, N., 2016. On the role of performance interference in consolidated environments. In: International Conference on Autonomic Computing (ICAC). IEEE/USENIX.
Rameshan, N., Navarro, L., Monte, E., Vlassov, V., 2014. Stay-away, protecting sensitive applications from performance interference. In: Proceedings of the 15th International Middleware Conference. ACM, pp. 301–312.
Shirvani, M.H., Ghojoghi, A., 2016. Server consolidation schemes in cloud computing environment: a review. European Journal of Engineering Research and Science 1 (3).
Silva, J., Boeres, C., Drummond, L., Pessoa, A.A., 2015. Memory aware load balance strategy on a parallel branch-and-bound application. Concurrency and Computation: Practice and Experience 27 (5), 1122–1144.
Steffenel, L.A., Martinasso, M., Trystram, D., 2007. Assessing contention effects on MPI_alltoall communications. In: International Conference on Grid and Pervasive Computing. Springer, pp. 424–435.
Sudevalayam, S., Kulkarni, P., 2013. Affinity-aware modeling of CPU usage with communicating virtual machines. Journal of Systems and Software 86 (10), 2627–2638.
Tierney, B., Kissel, E., Swany, M., Pouyoul, E., 2012. Efficient data transfer protocols for big data. In: 8th International Conference on E-Science (e-Science). IEEE, pp. 1–9.
Tsao, S., Chen, J.J., 2012. SEProf: a high-level software energy profiling tool for an embedded processor enabling power management functions. Journal of Systems and Software 85 (8), 1757–1769.
Tsuruoka, Y., 2016. Cloud computing-current status and future directions. Journal of Information Processing 24 (2), 183–194.
Vandekerckhove, J., Matzke, D., Wagenmakers, E., 2014. Model comparison and the principle of parsimony. In: The Oxford Handbook of Computational and Mathematical Psychology, pp. 300–317.
Wallshein, C.C., Loerch, A.G., 2015. Software cost estimating for CMMI level 5 developers. Journal of Systems and Software 105, 72–78.
Wijayasiriwardhane, T., Lai, R., 2010. Component point: asystem-level size measure for component-based software Ssystems. Journal of Systems and Software 83 (12), 2456–2470.
Xavier, M.G., Neves, M.V., Rossi, F.D., Ferreto, T.C., Lange, T., De Rose, C.A., 2013. Performance evaluation of container-based virtualization for high performance computing environments. In: 21st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP). IEEE, pp. 233–240.
Xu, C., Chen, X., Dick, R.P., Mao, Z.M., 2010. Cache contention and application performance prediction for multi-core systems. In: International Symposium on Performance Analysis of Systems & Software (ISPASS). IEEE, pp. 76–86.
Yokoyama, D.M.M., 2015. Modelo para o Escalonamento de Aplicações Científicas em Ambientes de Nuvens Baseado em Afinidade. Laboratório Nacional de Computação Científica Master's thesis.
Younge, A.J., Henschel, R., Brown, J.T., Von Laszewski, G., Qiu, J., Fox, G.C., 2011. Analysis of virtualization technologies for high performance computing environments. In: International Conference on Cloud Computing (CLOUD). IEEE, pp. 9–16.
Yu, S., Liu, H., Chen, Z.J., Hsieh, B., Shao, L., 2012. GPU-based parallel reservoir simulation for large-scale simulation problems. In: Europec/EAGE Annual Conference. Society of Petroleum Engineers (SPE).

Maicon Melo Alves is a Ph.D. student at the Institute of Computing at the Fluminense Federal University (UFF). He graduated in Computer Science from the Cândido Mendes University in 2004 and received his M.S. degree in Computer Science from the Federal University of Rio de Janeiro in 2012. He is employed by Petrobras, the brazilian oil state company, and is interested in High Performance Computing, cloud computing and parallel processing.

Lúcia Maria de Assumpção Drummond graduated in Mathematics in Computing Modality from the State University of Rio de Janeiro in 1987, received the M.S. degree in Systems Engineering and Computer Science from the Federal University of Rio de Janeiro in 1990 and the Ph.D. degree also in Systems Engineering and Computer Science from the Federal University of Rio de Janeiro in 1994. She is currently a Full Professor at the Institute of Computing of the Fluminense Federal University (UFF), engaged in the graduation and the graduate program, advising Master's and doctoral students. Her research interests include parallel processing, High Performance Computing and distributed algorithms.