The Journal of Systems and Software 84 (2011) 1270–1291

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

The effects of scheduling, workload type and consolidation scenarios on virtual machine performance and their prediction through optimized artificial neural networks

George Kousiouris [a,*], Tommaso Cucinotta [b], Theodora Varvarigou [a]

[a] Dept. of Electrical and Computer Engineering, National Technical University of Athens, 9, Heroon Polytechniou Str., 15773 Athens, Greece
[b] Real-Time Systems Laboratory (ReTiS), Scuola Superiore Sant'Anna, Via Moruzzi, 1, 56100 Pisa, Italy

ARTICLE INFO

Article history:
Received 25 February 2011
Received in revised form 6 April 2011
Accepted 6 April 2011
Available online 14 April 2011

Keywords:
Virtualization
Real-time scheduling
Cloud computing
Artificial neural networks
Genetic algorithms
Performance prediction

ABSTRACT

The aim of this paper is to study and predict the effect of a number of critical parameters on the performance of virtual machines (VMs). These parameters include allocation percentages, real-time scheduling decisions and co-placement of VMs when these are deployed concurrently on the same physical node, as dictated by the server consolidation trend and the recent advances in the Cloud computing systems. Different combinations of VM workload types are investigated in relation to the aforementioned factors in order to find the optimal allocation strategies. What is more, different levels of memory sharing are applied, based on the coupling of VMs to cores on a multi-core architecture. For all the aforementioned cases, the effect on the score of specific benchmarks running inside the VMs is measured. Finally, a black box method based on genetically optimized artificial neural networks is inserted in order to investigate the degradation prediction ability a priori of the execution and is compared to the linear regression method.

1. Introduction

During the recent years, server consolidation through the use of virtualization techniques and tools such as VMware (http://www.vmware.com/), XEN (http:www.xen.org/), KVM (http://www.linux-kvm.org/) is widely used in data centers or in new computing paradigms such as Cloud computing (Armbrust et al., 2010). Through this technique, applications with different characteristics and requirements can run inside virtual machines (VMs) on the same physical multi-core host, with increased levels of security, fault tolerance, isolation, dynamicity in resource allocation and ease of management.

However, a number of issues arise from this advancement in computing, like the degradation of the performance of the applications, due to the usage of the virtualization layer (Tikotekar et al., 2008). Except for the extra layer, a number of other parameters may also affect the performance in distributed and virtualized environments. These may be the percentage of CPU allocated to the VM and its effect on execution time and the scheduling granularity (the way this percentage is assigned) that may also affect the latter. What is more, consolidation decisions, whether these imply the assignment of VMs to the same core or neighbouring cores on the same physical node, may have an effect on application performance. In addition, different types of applications may have different interference effect on the co-scheduled ones on the same physical node. This may happen due to application internal characteristics, like type of calculations, usage of specific sub-circuits of the CPU, effect due to cache sharing, memory access patterns etc.

The degradation of performance in this case may lead to increased need for resources. However, the current Cloud business SPI model (Youseff et al., 2008) identifies discrete roles between the entity that offers a platform (PaaS provider) and the entity that offers the resources which this platform may use (IaaS or Cloud provider). This model, in conjunction with the aforementioned performance overhead, creates the following problem:

- A PaaS provider has estimated that *X* number of resources are needed for meeting the QoS features of the application inside the VM. This estimation will probably be made after observing the application behavior when running as standalone.
- The IaaS provider receives a request in order to reserve this *X* amount of resources. In an attempt to maximize profit, the VMs of the application will be consolidated with other VMs that are admitted in the infrastructure. The way this consolidation is performed is unknown to the PaaS layer. This does not mean that

* Corresponding author. Tel.: +30 6939354121.
E-mail addresses: gkousiou@mail.ntua.gr, gkousiou@yahoo.com (G. Kousiouris), tommaso.cucinotta@sssup.it (T. Cucinotta), dora@telecom.ntua.gr (T. Varvarigou).

0164-1212/$ – see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.jss.2011.04.013

the IaaS provider will give less resources. The same amount as requested will be allocated.
- The performance degradation due to the aforementioned consolidation and according interference will be translated to decreased Quality of Service (QoS) levels for the application.
- The breach of confidence between the PaaS and IaaS provider is inevitable. The former will consider that the latter has deceived him/her, and that the allocation of resources was less than agreed. The latter will consider that the former made a poor estimation and is trying to transfer the responsibility to the infrastructure layer.

Therefore, it is imperative for an IaaS provider to be able to predict accurately the overhead that is produced by the consolidation strategy and therefore increase a priori the number of resources allocated to an application accordingly. This will not only lead to improved reputation but will enable the IaaS provider to optimize the assignment of VMs to hosts or even cores of one host, depending on the type of workload these VMs produce and the estimated interference of one type of workload over the other. By knowing which combinations result in less overhead, over-provisioning with regard to standalone execution can be minimized.

The aim of this paper is to take under consideration a wide range of parameters that affect application performance when running on consolidated virtualized infrastructures and quantify this overhead. Therefore, IaaS providers may perform intelligent decisions regarding the placement of VMs on physical nodes. This will mitigate the aforementioned problem and will aid the IaaS layer to minimize the need for over-provisioning. As indicative applications we consider the 6 standard Matlab benchmark tests, due to the fact that they represent a wide area of computational workload, from mathematical calculations to graphics processing, and are easily reusable by other researchers. The following aspects are investigated:

- Scheduling decisions for EDF-based (Earliest Deadline First) scheduling (budget Q over period P), both in terms of CPU share assigned (Q/P) and granularity of this assignment (P). This type of scheduling (Liu and Layland, 1973), which is very popular in the real-time world, is used in order to guarantee that the assignments of CPU percentages to the VMs will be followed and to ensure temporal isolation.
- Different combinations of benchmark tests, in order to discover the ones that show less interference (and therefore overhead), when running concurrently on the same node.
- Different placement decisions and how these affect the overhead (placement on the same core, in adjacent or non-adjacent cores in one physical node, compared with standalone execution). As adjacent cores we consider the ones that share a L2 cache memory.
- Ability to predict in advance these effects in order to proactively provision the resources needed by an application and be able to choose the suitable configuration that creates the least overhead. For achieving this, artificial neural networks (ANNs) are used, that are automatically designed and optimized through an innovative evolutionary (GA-based) approach. The topology of the networks is dynamic in terms of number of hidden layers, neurons per layer and transfer functions per layer and is decided through the optimization process of the GA.

The remaining of the paper is structured as follows. In Section 2, similar approaches in the related field are presented, while in Section 3 an extended analysis is made on the chosen parameters for investigation. Section 4 contains the description of the test-bed and the measuring process, while Section 5 presents the analytical results from the experiments. Section 6 details the approach in order to predict in advance the effect and compares it to the multi-variate linear regression method, while Section 7 provides the overall conclusions from this study and intentions for the future.

2. Related work

In the past, we have investigated partly a number of the parameters that are considered here. In our previous works (Cucinotta et al., 2010a; Kousiouris et al., 2010), the main focus was given on the effect of scheduling parameters and % of cpu assignments on the QoS offered by specific applications (mainly interactive ones).

Given that virtualization technologies have become very popular during the recent years, a large number of interesting research efforts exist. In (Padala et al., 2007), the authors investigate the overhead that is inserted through the use of different virtualization tools, with the intention to discover the most efficient one. In (Makhija et al., 2006), the main focus is the scalability of virtual machines with mixed workloads and the effect on their performance when consolidated. The benchmarks are based on server applications. With the extended use of Cloud technologies, applications that are envisioned to be part of their workload may have more complicated workloads rather than traditional data center ones. Such examples may be the case of the Digital Film Post-production application (http://www.irmosproject.eu/Postproduction.aspx) of the IRMOS project or the availability of mathematical programming and optimization tools like GAMS in Cloud infrastructures (http://support.gams-software.com/doku.php?id=platform:aws). Furthermore, no temporal isolation between the virtual machines is used.

A very interesting approach, similar to our work, is Koh et al. (2007). In this case, the authors study the performance interference of combinations of elementary applications when running inside co-allocated VMs. Interesting score methodology is presented that can also be used for classifying applications. The workload characteristics of each application are gathered in order to be used as a comparison in the future for identifying unknown applications that are going to be inserted in the infrastructure and as predictors in a linear regression model. The main differences of our approach lie on the fact that we investigate a limited number of specific and generic benchmark tests, with the use of real time scheduling and a different prediction method based on ANNs. Especially the use of RT scheduling ensures the temporal isolation of VMs when running concurrently on the same CPU and the fair division of the resource percentages that each VM utilizes. In addition, we investigate different hardware setup interferences, such as running VMs on the same core or in adjacent cores in multi-core architectures and different setups in terms of granularity and percentage for each elementary application.

In Soror et al. (2008), a very promising approach for dynamically handling the resources on which concurrent VMs are executed is portrayed, in order to optimize system utilization and application response times. This is based on both offline recommendations and online corrections. However this work is centered around database applications. In Moller (2007), a design and validation methodology of benchmarks for virtual machines is presented, where also the concurrency effects are investigated. In Tickoo et al. (2010), a number of configurations is taken under investigation, like running on the same or adjacent cores, through hyper-threading or not. Experiments are focused on 3 tests and without taking under consideration scheduling parameters. Prediction methods are based on analytical formulas.

In Jin et al. (2008), VSCBenchmark is described, an analysis of the dynamic performance of VMs when new VMs are deployed. The authors investigate the behavior of the co-scheduled VMs for

both launching and maintaining multiple instances on the same server. Boot time for a VM is a very resource consuming task, that may lead other co-scheduled tasks to resource starvation. In our work, this is avoided through the use of the real time scheduling that limits the resources assigned to a task (like a VM) to a predefined percentage. Thus temporal isolation is achieved between the concurrently running tasks.

In Weng et al. (2009), an interesting approach for scheduling VMs is presented, with two modes of operation (high-throughput and concurrent) which aims at efficiently handling CPU time for the phases where the VMs have multiple threads running concurrently. This approach is more focused on the scheduling aspects and does not take into consideration concurrent VMs running on the same server. The VirtualGEMS simulator for testing different configurations with regard to L2 setup between different concurrently running VMs appears in García-Guirodo et al. (2009).

A benchmarking suite specifically for VM performance is depicted in El-Refaey and Rizkae (2010). While it pinpoints some important features for benchmarking of virtualized applications, it mainly focuses on one type of workload (linux kernel compilation). A survey of virtualization technologies in addition to performance results for different configurations appears in White and Pilbeam (2010). In Cucinotta et al. (2009, 2010b), an investigation is carried out on the possibility to enhance the temporal isolation among virtual machines concurrently running on the same core, by using the IRMOS real-time scheduler, focusing on compute-intensive and network-intensive workloads.

In Khalid et al. (2009) a very interesting approach for adaptive real-time management of VMs in High Performance Computing (HPC) workloads is presented. This mechanism is used during the operation of the HPC infrastructure, in which the VMs have a predefined running period which is decided by the infrastructure sharing policy. The main goal is to calculate in real time if the job that is included in the VM is going to be finished in time or not. The ones that will not anyway finish in time are prematurely killed in order to improve the success rate of the overall infrastructure. In our view, this work can be used in combination with the one presented in this paper. The models that are described here can be used for observing and minimizing the overhead of a specific allocation over the different nodes, while the work presented in Khalid et al. (2009) can be used for the runtime optimization of the virtualized infrastructure.

GA-based optimization of ANNs has been used in the past in a number of significant works. For example, in Leung et al. (2003), a number of the parameters of the network (like connection weights) are chosen through GAs. The main disadvantage of this approach lies on the fact that a static structure for the network is required. In other cases (like in Ilonen et al. (2003)) they are used for indirect optimization, through for example optimizing the training process. In Fiszelew et al. (2007), another approach combining GAs and ANN architecture is presented, however the structure of the network is limited to two hidden layers and static transfer functions, while experimenting with a different set of parameters such as number of nodes and their connections. The same applies for Giustolisi and Simeone (2006). In our approach, the networks may have a dynamic topology, in terms of number of hidden layers, number of neurons per layer and transfer functions per layer. Other parameters such as different training methods can be added at will.

3. Investigated parameters

In a general purpose operating system, when two tasks start to execute, they compete with each other for the underlying computational resources. The amount of the latter that is assigned to each task may be divided according to the temporal needs and demands of them. However, this results in an inability to predict the QoS offered by an application running inside a VM, in shared infrastructures. The percentage of CPU allocated to it may vary significantly depending on the co-located tasks demand (like the booting of another VM in Tickoo et al. (2010)). One solution to mitigate this effect is derived from the real time world. Through the use of EDF-based scheduling, the percentage assigned to a task may be guaranteed. In general, for this type of scheduling, there are two significant parameters, the budget Q and the period P. Q is the computational time on the physical node that is assigned to a task over a period P. Thus, the ratio of Q/P is the resulting percentage on the machine. However, for a given percentage, different granularities may exist, depending on P (for example, 50% may mean 50 ms every 100 ms or it may mean 250 over 500 ms). One interesting question is how these parameters affect application performance.

As investigated in Cucinotta et al. (2010a), Kousiouris et al. (2010), the effect of real-time scheduling parameters may affect the application performance in a variety of ways. For example, in our aforementioned works, that investigate interactive applications, different granularities of the same CPU percentage assignments lead to increased standard deviation values for server response times. The conclusion from that analysis was that for interactive applications, low granularity (small scheduling period P) leads to more stable performance for the same percentage of CPU assigned. In this work, we focus on 6 computationally intensive applications. The effect of scheduling decisions (for both percentage of CPU allocated and granularity of this allocation) on their performance metrics is observed along with the interference of the co-allocation of VMs for the same CPU % assigned.

As identified in IRMOS project D5.1.1 (2009), the metric that is currently used by infrastructure providers in order to describe hardware capabilities is mainly CPU frequency. However this metric is far from sufficient, given that it does not reflect the physical node's ability to solve problems (does not take under consideration other factors such as bus speed, memory speed, etc.). Other approaches such as the Berkeley dwarfs (http://view.eecs.berkeley.edu/wiki/Dwarfs) or Matlab benchmark tests are more able to capture a node's computational capability. In this paper the second approach was chosen. The Matlab benchmarks (http://www.mathworks.com/help/techdoc/ref/bench.html) consist of 6 tests, that measure the physical node's (and Matlab's for this matter) ability to solve problems with different computational requirements. Therefore these tests are used for both determining the hardware computational capability (test score) and the characterization of the types of workloads (test number).

What is more, they represent a wide area of types of workloads (Table 1) with a limited number of tests and it is the reason the specific approach was chosen. For these, we investigate the effect of the CPU allocation and the effect of their combination in concurrently running VMs. It is expected that different combinations will have different effect, due to the nature of the computations. The degradation of the performance is reflected on the test score. Lower test scores indicate better performance. In general, the test score is the time needed to make the necessary calculations for each test.

Furthermore, different allocation scenarios are investigated for multi-core architectures. The VMs are pinned with a variety of ways

Table 1
Details regarding the Matlab benchmark tests and the specific computational patterns that they represent (http://www.mathworks.com/help/techdoc/ref/bench.html).

Test 1	Floating point operations with regular memory accesses
Test 2	Floating point operations with irregular memory accesses
Test 3	Data structures
Test 4	Floating point and mixed integer operations
Test 5	2-D graphics
Test 6	3-D graphics

on cores of the same physical node in order to examine the interference effect per scenario on their performance. This is expected to be reduced due to the temporal isolation when sharing the same core (with the exception of cache influence). However it is interesting to see what happens when the VMs that contain the individual tests are scheduled on different cores on the same physical node, thus running in real parallelism and contending for the same sub-circuits like the RAM bus.

So, overall what is investigated is:

- different core% assignments for a single VM running an elementary test,
- different granularity of allocation assignments for a single VM running a test,
- mutual interference effect of combination of tests for concurrently running VMs on the same core (shared L1 and L2 cache memories) and for a variety of scheduling periods,
- mutual interference effect of combination of tests for concurrently running VMs on adjacent cores (with shared L2 cache) on the same physical node and for a variety of scheduling periods,
- mutual interference effect of combination of tests for concurrently running VMs on non-adjacent cores (non-shared L1/L2 cache) on the same physical node and for a variety of scheduling periods,
- ping times for different ping periods, %core assignments and scheduling periods of VMs in order to investigate the effect on network responsiveness.

4. Testbed

For the real time scheduler, the one that is presented in (Checconi et al., 2009) was chosen. The host of the experiments was a quad core (2.4 GHz) CPU, with 8 GB of RAM and 8 MB of L2 cache (divided in two 4 MB parts per core couple). Power management was switched off in order not to dynamically change the processor speed for power efficient operation. The host OS is Ubuntu Linux 10.10, the VM OS is Windows 7 (with Cygwin and OpenSSH) and the hypervisor is KVM. The Matlab executable containing the tests took as arguments the duration of the experiment, the number of test to run, an execution ID and the scheduling parameters (Q, P) for documentation purposes. It was compiled in Matlab R2007b. In order to be able to run individual tests and not the entire suite, the variation of the benchmark function that is provided in (http://www.mathworks.com/matlabcentral/fileexchange/11984) was used. This was combined with suitable logic in order to perform the tests for a specific period of time and not for a specific number of runs. The reason for this is that each one of the 6 elementary tests has a different execution time, which is also affected by the resources allocated to it. So the synchronization of them cannot be based on number of runs. Furthermore, in order to run each test for an increased number of times, this duration was significantly larger so that a large sample could be collected (in the range of hundreds of test runs per configuration). From this sample, the mean value of the execution times was extracted for each case.

The experiments were coordinated by a Java module (Coordinator) that was pinned on one core (CPU 0). This code implemented the interface towards the real time scheduler, in order to change the Q, P scheduling parameters on the processes of the VMs. The VMs were pinned to either one core (CPU 2) or each one on different cores (CPU 1, 2 or 3), depending on the experiment. After the setup of the scheduler, the Coordinator was used in order to connect into the VMs through SSH and launch different combinations of the Matlab executables for a specific period of time. The time was measured from inside Matlab, and each execution with a given configuration was performed until a certain time period had passed (500 s). This leads to a small violation of the time period, but it was much smaller than the overall duration. It is critical to mention that if the functions used to measure time are based on the clock function of Matlab, this is far from accurate. It was observed that with high utilization inside the VM, these timing mechanisms lost completely track of time (indicatively, while the simulation was running for 2 h, inside the VM only 5 measured minutes had passed). This is why the tic-toc measuring function of Matlab was used.

The pseudo-code for the Coordinator appears in Fig. 1 and for the executable inside the VM in Fig. 2. The overall implementation design and the interactions between the necessary software components appear in Fig. 3. In detail, the different steps needed are:

- Launching of the Java Coordinator. This component has the responsibility of keeping track of the execution loops for the various parameters of the experiments like scheduling periods, CPU percentages assigned and test combinations. Furthermore, it raises the threads that contact the scheduler interface (Python script) for setting up the CPU allocations for each VM.

```
1. for different combinations of P
2.             for different combinations of Q/P
3.                             set scheduling params for each VM
4.                             for all unique combinations of tests
5.                                     raise threads to connect and execute
6.                                     tests inside the VMs
7.                                     synchronize threads
8.                             end
9.             end
10.end
11.gather results from VMs
12.merge results based on execution ID
```

Fig. 1. Pseudo-code for Java Coordinator sequence of actions.

```
1. Take input arguments (runDuration, test number X, execution ID)
2. Measure time (tic)
3. while (duration< testDuration)
                      Run Test X (1 time)
                      Add result to array
                      Measure time (toc)
        End
4. Extract statistical data (mean, std dev) from the individual values
5. Enter log data
```

Fig. 2. Pseudo-code for Matlab executables inside the VMs.

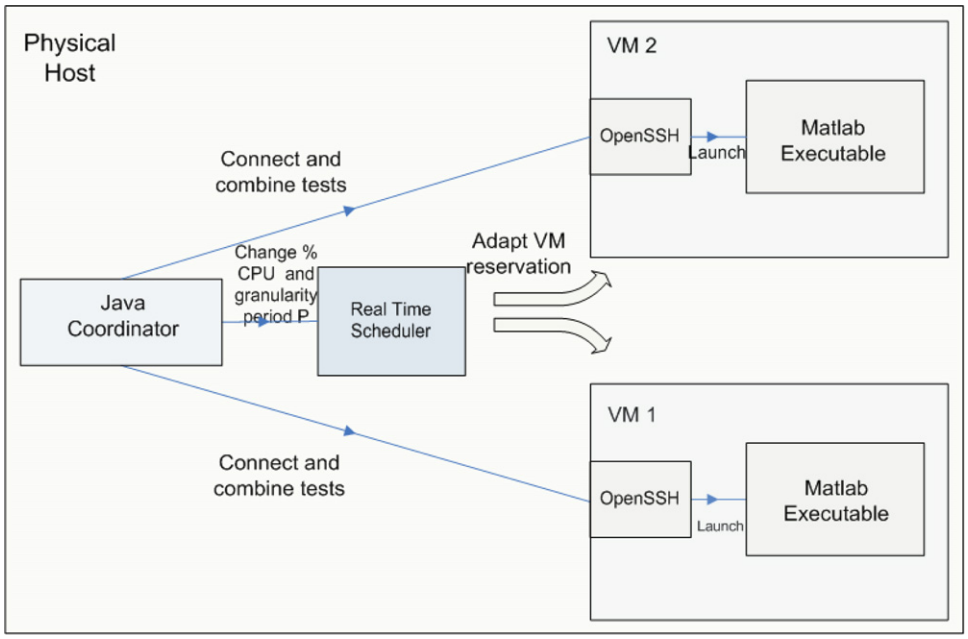


Fig. 3. Software components needed for the implementation and their interaction: the main component is the Java Coordinator which keeps track of execution loops in order to cover the investigated interval, sets up the scheduling parameters of the VMs and connects to the guest OS in order to launch the specific test combination dictated by the current execution.

- The next step is to raise different threads in order to connect to the VMs and launch the Matlab executables with the specific arguments (test combinations). These threads are synchronized in the end of their execution, in order to ensure the concurrency of the next combination execution. It is critical to mention that in order to ensure a stable execution of the threads for an unlimited period of time, the implementation suggested in (http://www.javaworld.com/javaworld/jw-12-2000/jw-1229-traps.html?page=4) was followed. This guarantees that the output and error streams are captured efficiently, in order to avoid computer hangs.
- When both executions inside the VMs are completed, the Java Coordinator progresses with the next configuration dictated by the loop.

Thread synchronization was performed in order to ensure that the VMs are executed concurrently. The results of each VM were kept in a logfile locally at the VM. At the end they were merged in order to obtain the overall logs. Details regarding the testbed are documented in Table 2.

The duration of each run (a specific test combination with a specific hardware configuration) was set to 500 s. This was sufficient time to ensure that the threads would be concurrently executed for the majority of the time, without having random ssh connection delays influence this process. The ssh connection delays ranged from 2.8 to 3.8 s. This is why thread synchronization was performed in the end of the 500-s interval. The thread that finished first waited for the second one to complete. It was also enough in order to let the tests run for an increased number of runs (in the range of hundreds), thus leading to sufficient sample gathering.

Table 2
Test-bed details regarding hardware and software.

Host OS	Linux Ubuntu 10.10
Guest OS	Windows 7
Hypervisor	KVM
Benchmarks	Matlab R2007b executable
Physical host	Intel Core 2 Duo Q6600 (Quad core 2.4 GHz, 8 MB cache, 8 GB RAM)

Furthermore, an execution ID was passed inside the VMs, in order to be included in the reported results. This was a unique number for each examined combination. This is necessary due to the fact that it was observed that for a number of random reasons (like broken tcp pipes, etc.) a limited number of combinations did not succeed in starting or finishing. For these cases the ID did not get logged. This way the results from the separate files that are kept inside each VM and consolidated in the end could be filtered in order to rule out the aforementioned cases and repeat the experiment for them.

5. Measurements

In the following paragraphs the different deployment scenarios are described, along with the resulting measurements. For all cases, the scheduler granularity was investigated from 200 to 800 ms. A warm-up period for the system was introduced, as suggested by the literature, in order for the VMs to behave in a more stable fashion. That is why the experiments began from 150 ms, but the results were discarded for that specific value. Each data point in the graphs shown in the next sections is the mean value of the individual test runs inside a 500-s interval. In general these individual test runs were in the range of hundreds of times.

5.1. Single VM on a core

In this scenario the maximum percentage assigned to each test is 80%. This is due to the fact that the remaining part is left for host system tasks to utilize. In this configuration, each VM is executed as standalone on one core. No other VM is running. This was performed in order to see the effect of the granularity of the scheduling parameters and the percentage allocation, in addition to acquiring the baseline scores of un-interfered execution (standalone) for comparison with the VM consolidation scenarios. As scheduling parameters are considered the CPU share X assigned to a task in a period P. In every period, the task is given X^*P ms of CPU time. For example if we have a scheduling period of 500 ms and a CPU share of 40%, this means that the task will be given 200 ms of CPU time every 500 ms.

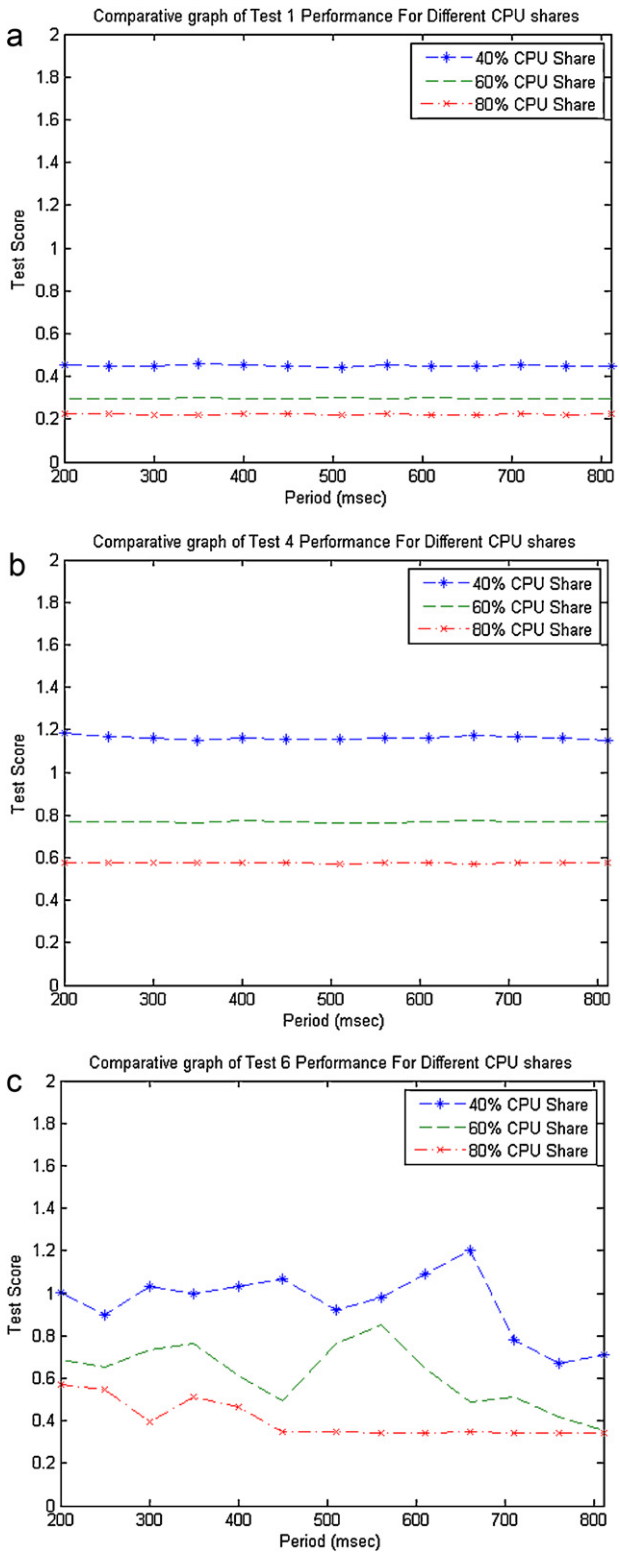


Fig. 4. Test scores for different CPU percentages and granularity (scheduling period) when the tests are running without any interference (co-scheduled tests). The test score (vertical axis) indicates the time needed for executing one test run and has been calculated from the mean value of hundreds of test runs inside a 500-s execution run. The scheduling period (horizontal axis) is the time interval in which the allocation of *X*% CPU share is guaranteed. (a) Standalone test 1; (b) Standalone test 4; (c) Standalone test 6.

In Fig. 4 the scores for each test are portrayed, for different CPU shares and periods of assignment of them. Lower test scores indicate better performance. In essence, the test score is the time needed for completing one test run. Only an indicative number is portrayed here, remaining graphs are included in Appendix A (Fig. A.1).

For tests 1–4, the conclusion is that the behavior is independent of the granularity of scheduling. This is anticipated since in a standalone basis, the effect of interference in cache memories from task switching is non-existent. With relation to the improvement of performance when the CPU share increases, this is almost linear, as seen from the inversely proportional test scores.

For tests 5–6 (graphics tests), the behavior is more oscillatory, which can be attributed to the difficulty of the virtualization layer to access or emulate the graphics card.

5.2. Concurrent VMs collocated on the same core

The second experiment aims at examining the effect of concurrency on the performance of VMs that are running on the same core, thus using shared L1 and L2 cache memories. For this reason, each VM is allocated 40% of the core capacity. The performance scores are compared to the 40% scores of experiment A. Results are shown in Fig. 5. These illustrate the scores of each individual test, when running on the same core with each of the other tests. That is why the (i,j) figure does not coincide with the (j,i) figure. Lower test scores mean better performance.

It is evident from the above graphs that while the scheduling period *P* of the assignments increases, the performance overhead of the co-scheduled tests improves. This is owed mainly to the fact that with larger periods (and the same % of CPU) the applications have more time to utilize components such as the cache memories. A lower period results in more frequent task switching on the CPU and thus higher cache interference and contamination between the concurrent tasks. Furthermore, the overhead from the context switching (scheduler overhead) is increased.

Another conclusion was that in some parts the graphs seemed to portray some anomalies like in the test 3 graph (Fig. 5c). However, after carefully observing the results, it seemed that when the high values of one test occurred, there were low values for the other test. This led us to plot the combined performance (added benchmark scores) of both co-scheduled VMs, which showed a more logical system-level graph. It seems that what matters in this case is which task starts first. In the test-bed description in Section 4 it was stated that the execution threads are synchronized in the end of the testing period, because this cannot happen in the beginning. The threads are launched simultaneously, but due to a number of random reasons (like ssh delay for example), the execution of each test in the respective VMs does not start at exactly the same time. This delay was in the range of 3 s, however it was much lower than the selected 500 s duration of each combination. From the system level graphs more accurate conclusions can be deducted. These are combined with the other scenarios and are depicted in Fig. 6.

Furthermore, there seem to be extremely high values for the 250 ms period *P*. This happens for almost all graphs and is contradictory to the general trend of the graphs. By taking under consideration the way the for loops are constructed in Fig. 1, we can conclude that this happens due to some internal interference in the guest OS (e.g. an OS task that starts executing inside the VM) and influences the measurements taken at this period.

5.3. Concurrent VMs collocated on adjacent cores

The third experiment aims at investigating the effect on the performance of VMs when running on different but adjacent cores on the same CPU. This is to identify the effect of

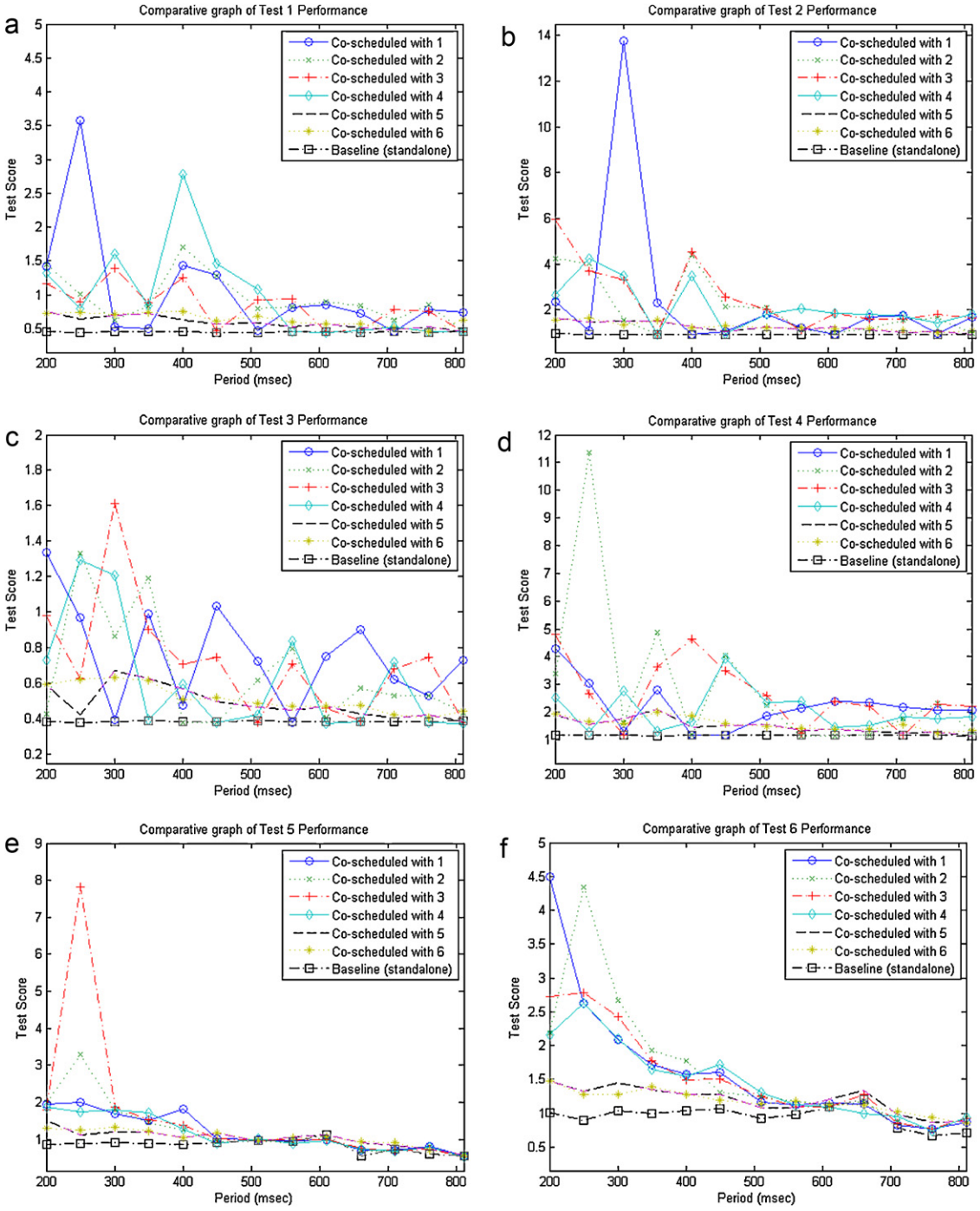


Fig. 5. Effect on individual test scores for different combinations of concurrent tests. Each of the graphs indicates a test's performance. It contains 7 lines which correspond to this test's performance, when each of the other tests is running on the other VM, in addition to its performance when it is executed as standalone for comparison purposes. (a) Test 1 performance; (b) test 2 performance; (c) test 3 performance; (d) test 4 performance; (e) test 5 performance; (f) test 6 performance.

L1 cache memory interference, when compared to the results of B. In this setup, the cores have different L1 cache memories but common L2 ones. The chosen utilization percentage for this experiment was 40% per core. The reason for choosing this percentage was to have comparable results with B, for which maximum assignment of two tasks on one core is 40% per task (after subtracting approximately 20% that is always available for system tasks). Comparison with the other scenarios is included in Fig. 6.

5.4. Concurrent VMs collocated on non-adjacent cores

The fourth configuration was decided to be pinning the VMs on non-adjacent cores. The reason for this setup is that the physical node of the experiments has 4 cores, which share a L2 cache in pairs. So by pinning the VMs on non-adjacent cores, the gain in terms of having no shared cache memories is investigated, in comparison to the previous experiment where L2 memory was shared. For all scenarios, the combined performance of the VMs per com-

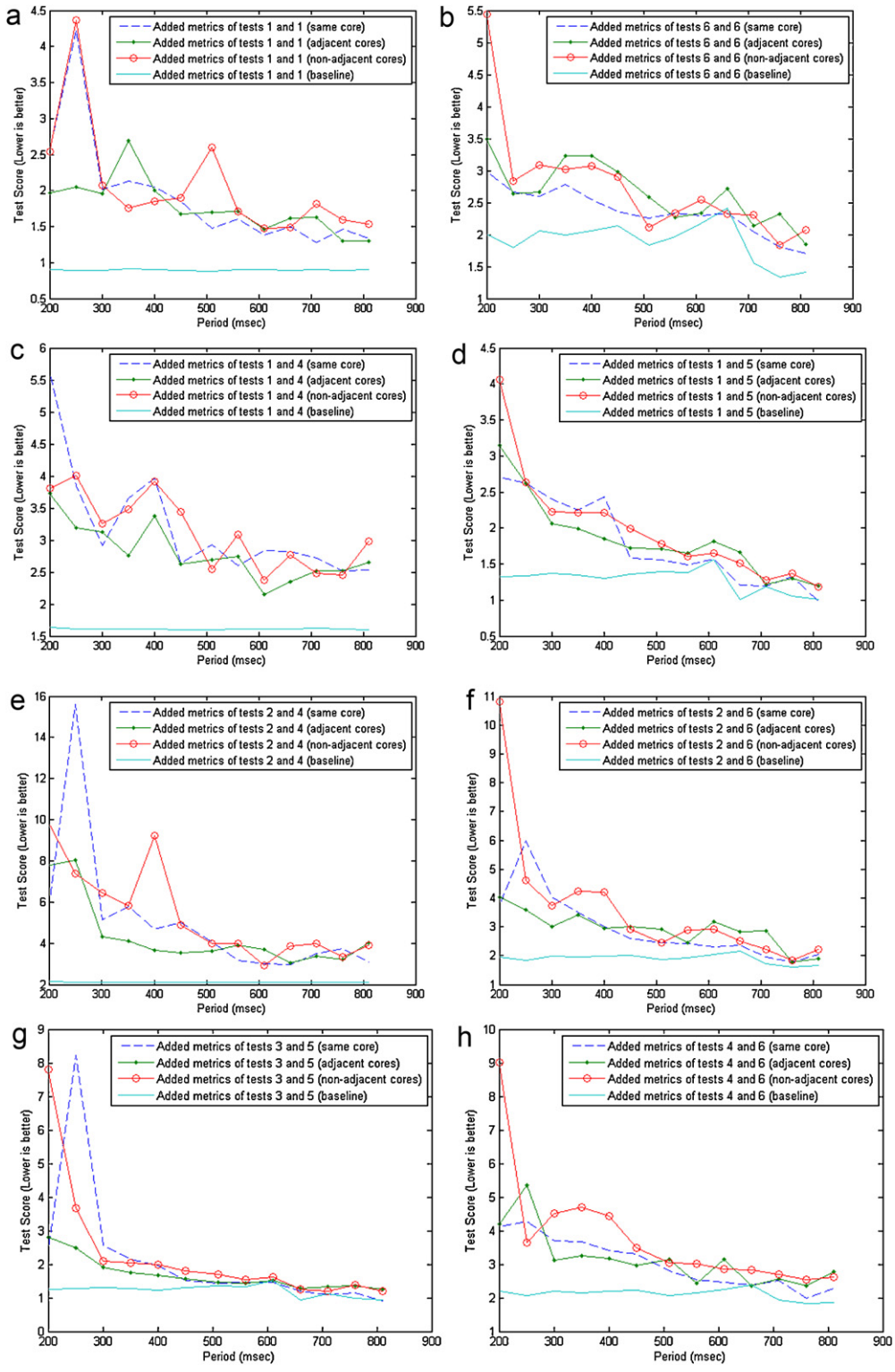


Fig. 6. Comparison of system level (added scores) performance for all deployment scenarios and 40% CPU allocation per VM. In each of the graphs a specific combination of tests is investigated in order to compare their added scores (vertical axis) with regard to different deployment scenarios and how this performance is affected by the scheduling period *P* (horizontal axis). (a) Tests 1 and 1 combined performance; (b) tests 6 and 6 combined performance; (c) tests 1 and 4 combined performance; (d) tests 1 and 5 combined performance; (e) tests 2 and 4 combined performance; (f) tests 2 and 6 combined performance; (g) tests 3 and 5 combined performance; (h) tests 4 and 6 combined performance.

bination of tests is depicted in Fig. 6. In this figure, we have added the test scores of both VMs and have plotted all the deployment scenarios for comparison purposes. The remaining combinations that are not included in this figure are located in Appendix B (Fig. B.1).

The anticipation during the beginning of these experiments was that interference between memory usage of concurrently running VMs would significantly affect VM performance. This was validated from all tested cases. The baseline scores are much better than any other configuration, even though the percentage of core assigned to

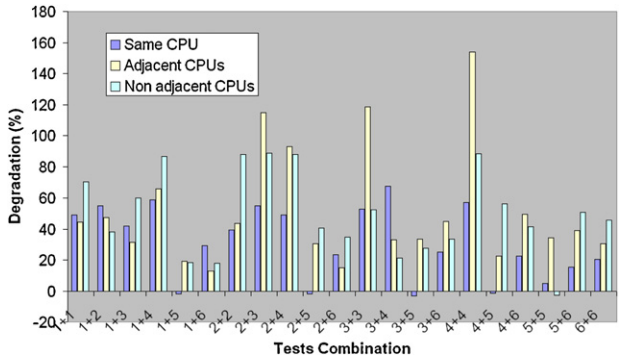


Fig. 7. Percentage of the degradation of benchmark scores for all test combinations and scenarios and $P = 800$ ms in order to illustrate the range of the overhead and the identification of optimum combinations. In the horizontal axis the different test combinations are portrayed while in the vertical one the degradation of the performance as extracted from Eq. (1).

the VMs was the same. Furthermore, different tests create different levels of interference, thus validating the initial assumptions that careful analysis must be made in order to optimize VM groupings on available nodes. Higher periods for the scheduler seem also to benefit the performance, due to better cache utilization.

However, it was also anticipated that moving from tightly coupled configurations (same core) to more loosely coupled ones (adjacent and non-adjacent cores) would benefit the performance of the VMs. After observing the graphs, we can conclude to the fact that this does not happen. Given that the loosely coupled cases mean L1 cache independence (for adjacent cores) and L1/L2 cache independence (for non-adjacent cores) it can be assumed that the bottleneck that is causing this lack of performance improvement has to do with RAM accesses. The factor of influence in this case can be the memory bus (FSB) that is necessary for accessing the RAM. In the observed hardware, this was a 1066 MHz bus. However, this bandwidth is shared between the different cores trying to access the main memory. If more than one cores try to access the RAM at the same time, the available bandwidth will be divided between them. Thus, when allocating VMs on different cores (with the same 40% share), the divided bandwidth is a limiting factor in comparison to the same core scenario. For the same core deployment, VMs are executed one after the other, in a sequential, pseudo-parallel fashion. During this time, each VM has all the available bandwidth of the FSB. When allocated in different cores, for the interval in which the 40% activation periods of the VMs overlap, their bandwidth on the FSB will be divided. Given that the number of accesses to RAM (or when) that are necessary can be fluctuating, this can explain why in some cases one configuration is better than the other, with one effect prevailing over the other (cache interference over shared FSB bandwidth).

In order to better conclude and illustrate for which combination of tests this improvement is greater, the percentage degradation of the system level test scores is portrayed in Fig. 7, for $P = 800$ ms. We chose the specific period due to the fact that it was shown from the previous graphs that for this value of P the system had the least overhead. The degradation of the performance is the ratio between the test scores of the added concurrent executions and the ones of the added standalone ones, for all the combinations of tests and is derived from Eq. (1).

$$\%\text{Degradation} = 100 \times \frac{(T_{A\_I} + T_{B\_I}) - (T_{A\_Base} + T_{B\_Base})}{T_{A\_Base} + T_{B\_Base}} \quad (1)$$

In this formula, A and B are the test numbers per combination, I is the deployment scenario (same core, adjacent cores, non-adjacent cores) and *Base* is the scores when the tests are executed as standalone, without any other test running on the physical node.

The aim of this figure is to show the concrete percentage overheads of different scenarios and for different test combinations and to identify any patterns that can be of benefit for extracting generic conclusions. From this it can be shown that the computational capabilities of the VMs can range from almost 0 and up to 160%, depending on the collocated tasks and the deployment scenario.

From this graph it is also evident that the best combinations are for all applications with test 5 and the second fittest candidate is test 6. The reason why these graphics tests are the fittest can be attributed to the fact that they use a small amount of data, on which they perform a large number of computations. Worst candidate for co-allocation is test 4, while the adjacent cores placement seems to produce the maximum overhead (combination of L2 and FSB bus interference). This is evidence of the importance of the investigation of the interference caused between co-scheduled VMs, which can lead to significant degradation of the QoS features of applications running in virtualized infrastructures. Furthermore, in most cases the adjacent core setup has the worst performance, due to L2 cache interference and division of memory bus bandwidth. L1 cache interference seems to affect only a limited number of combinations (like 1 and 2).

5.5. Statistical analysis of the measurements

In order to gather the dataset that was presented in the previous graphs, each configuration (specific scheduling period, specific test combination and deployment scenario) was let to run for 500 s. This means that each point in the graphs (with the according setup) was produced from taking the mean from all the runs executed in this 500-s interval. The number of runs was fluctuating due to the fact that each test had different computational needs and also the overhead from the concurrently running VMs was varying. However it was in the range of hundreds of times for each data point This is indicative also of the test scores, which indicate the time needed to perform one run. The mean value from these hundreds of times of executions for each configuration was used in the graphs. Details regarding the specific numbers for each experiment are included in Appendix D (Fig. D.1).

In order to further validate the data gathering process, we repeated a number of the experiments for 10 times. This means that the executions with a specific configuration (test combinations, Q/P ratio, P scheduling period) were executed 10 times in order to investigate the difference between non-consecutive executions with the same conditions. During this experiment, each configuration (500-s interval) was run for 10 times in a non-consecutive manner. The results (mean values of the test scores in the 500-s interval and standard deviations of the values inside this interval) are portrayed in Fig. 8. From this it can be observed that the difference between the different executions is very small.

Furthermore, the number of runs for each test in each of the executions was documented and is shown in Fig. 9. From this it can be observed that for each execution (from which the mean value and standard deviation that is portrayed in Fig. 8 was extracted) around 320 runs were conducted for test 4 and around 230 runs for test 5. The difference in the number of runs between the two tests occurs from the fact that each test has different types of computations, thus different time of execution. It was chosen to synchronize the executions based on a given time interval (and not on number of runs) in order to ensure the concurrency of the computations. By having the same number of runs (and different time for each run of each test) we would have the problem of one test finishing faster than another. Thus the remaining runs of the slower test would be executed as in the standalone phase.

From the above graphs we can conclude that due to the fact that the test runs in each execution are significantly high, repetition of

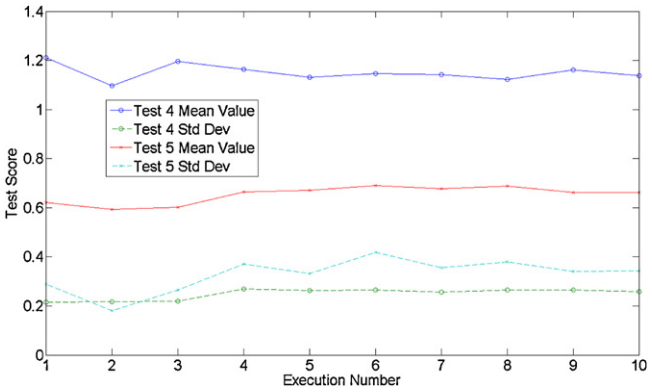


Fig. 8. Variation of the mean times and standard deviations of the test scores for 10 different non-consecutive executions of the same configuration (combination of tests 4 and 5, scheduling period = 700 ms, same core execution). The sample for extracting the mean value and std dev for each execution has been gathered after letting each execution run for 500 s.

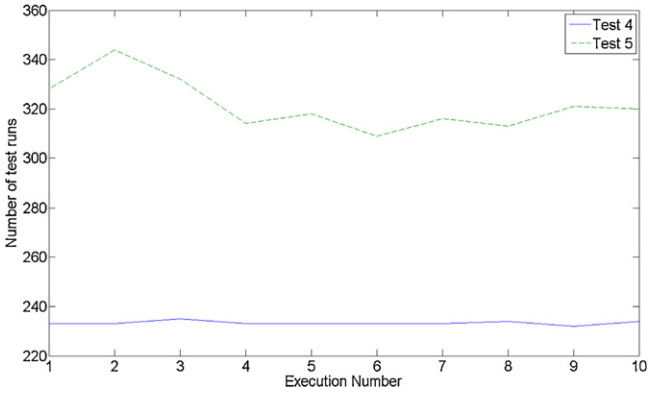


Fig. 9. Number of test runs inside the 500-s sample gathering interval for each of the different non-consecutive executions of the same configuration (combination of tests 4 and 5, scheduling period = 700 ms, same core execution).

the experiments will produce very similar values (as depicted in Fig. 8). The distribution of the individual values of the test scores for test 4 appears in Fig. 10, for one of the 10 executions (Fig. 10a) and all 10 repetitions (Fig. 10b). From this comparison it can be concluded that the two distributions are almost alike, showing the peaks in the same bins (0.8–1.2 and 1.2–1.4 values).

Regarding the statistical analysis of the samples gathered inside a single execution, the confidence intervals are depicted in Fig. 11, for both tests (4 and 5) of the selected run. For this run (500-s interval), test4 was executed 232 times while test 5 was executed 321 times. In order to process the results, the dfittool of Matlab was used. From this we can observe that, especially for test 5 there are two major areas of concentration of the test scores, around the 0.9 and 1.3 values. This can be attributed to the inactive periods inserted by the scheduler for the task. This configuration had a 40% share over a period of 700 ms. This means that for 60% of the time (420 ms) the task is deactivated. Thus if the test run manages to finish before the inactive period it will not have the extra delay of the 420 ms. If not, then it will finish right after that interval. This agrees also with the difference between the two major values of the distribution.

5.6. Network response

During the initial tests for the experiments of 5.1 it was noticed that for low values of period P (below 100 ms) the VMs were extremely unresponsive, even for a simple ssh connection or ping.

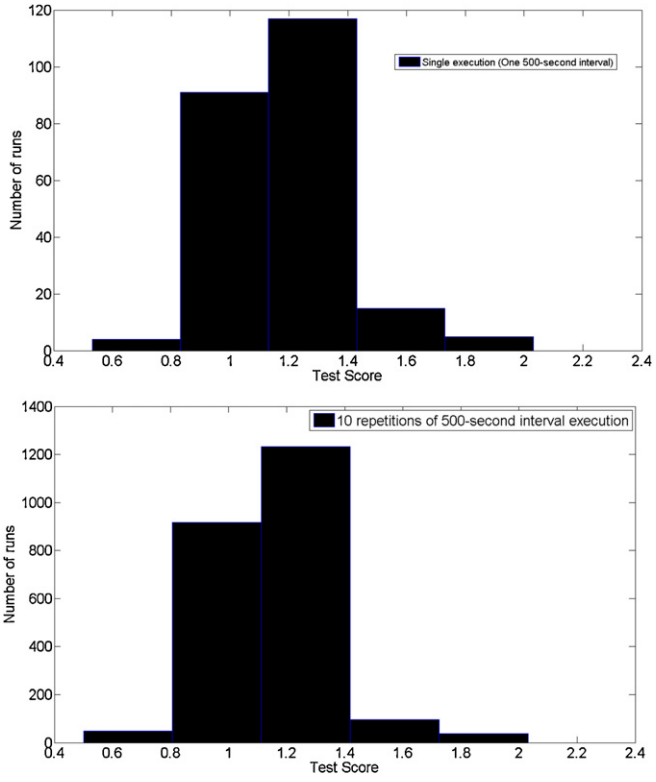


Fig. 10. Histogram of the distribution of individual times for each test run for (a) 1 execution of the 500-s interval and (b) 10 executions of the 500-s interval with a given configuration (combination of tests 4 and 5, scheduling period = 700 ms, same core execution).

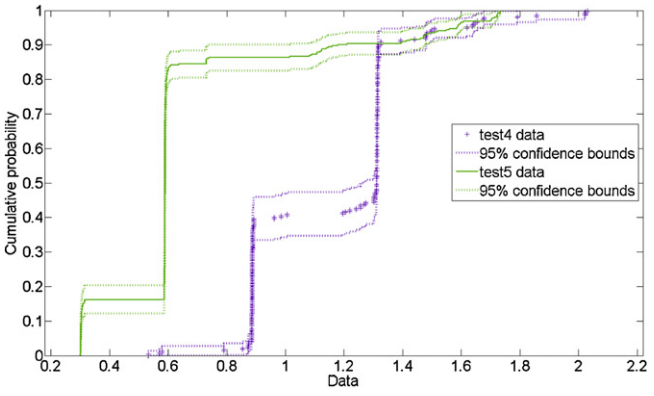


Fig. 11. Cumulative distribution function and confidence intervals for tests 4 and 5 execution samples for scheduling period = 700 ms, same core execution. The horizontal axis symbolizes the test scores and the vertical axis the probability to have these scores.

For this reason it was decided to conduct another experiment in order to investigate the deterioration of VM network response times. This investigation involved the pinging of one VM for a certain period of time (300 s) for a number of different configurations. Different ping periods were used. The number of samples can be derived from the division of the 300-s period with the pinging period. Different percentages of CPU assignments were also taken under consideration and different periods P of the scheduling allocation. Detailed values appear in Table 3.

In order to stress the CPU to full load, two approaches were considered. The first was with one of the Matlab benchmarks. In order to avoid the effect of some specific computation interference that could be caused by these tests (like increased I/O that could increase

Table 3
Parameters varied for the ping experiment.

Pinging period	0.5, 1, 1.5 and 2 s
Scheduling period *P*	150–800 ms (50 ms step)
VM status	Full load
Q/*P*	20, 40, 60 and 80%

the served interrupts and affect the ping responses), another executable was created. This was comprised of a simple infinite for loop that calculated the square value of the loop index. The results were almost identical. In the following graphs (Fig. 12), the mean value and deviation of the response times are depicted, for the most stressful experiment (pinging period = 0.5 s). More combinations are included in Appendix C (Fig. C.1).

It is imperative to stress that the test began from *P* = 50 ms of granularity, however the VMs showed little responsiveness, with many packet time outs occurring. Full responsiveness was shown from 150 ms values, and from these values and on the plots were created. What is seen from these measurements is that after the initial granularities when the VM is unresponsive, low values of *P* are more beneficial for the network metrics (both mean value and deviation from that for each individual packet time). Furthermore, one theoretical expectancy would be that the ping times would deteriorate as the *P* values increased, due to the fact that for low percentages of CPU assignments, large inactive periods of the task would affect especially the deviation times. However, as seen from the graphs, the highest values for both metrics are for middle cases of *P* (for the 450 ms case). After this maximum, the network metrics seem to improve again, however not to the levels of the low *P* values. Similar behavior was noticed for adjacent values of *P* (200 ms, 400 ms and 800 ms). These were skipped for better visibility of the graphs. Another interesting conclusion is that the pinging period does not seem to affect this behavior.

The similarity of the metrics for high values of the allocation *Q*/*P* (%CPU share) is due to the fact that for percentages close to 100% (like the 80% case), the CPU is almost dedicated to the task, so the period of this allocation does not influence the VM behavior. Almost the entire time the task is active.

For the statistical analysis of the measurements we have used as an example the configuration for scheduling period of 150 ms, pinging period of 0.5 s and CPU share of 60%. The distribution of the 600 individual values collected during the sample gathering experiment is depicted in Fig. 13a. In Fig. 13b the fitting of an exponential distribution to these values is shown, with mean value = 461.544, variance = 213,023, std error = 18.8739 and estimated covariance = 356.226. Exponential distributions are commonly used for networking times, as also shown in Cucinotta et al. (2010a). The CDF and confidence intervals for all CPU shares of this configuration are shown in Fig. 14, from which the benefit of having more CPU share is visualized in a more quantitative way.

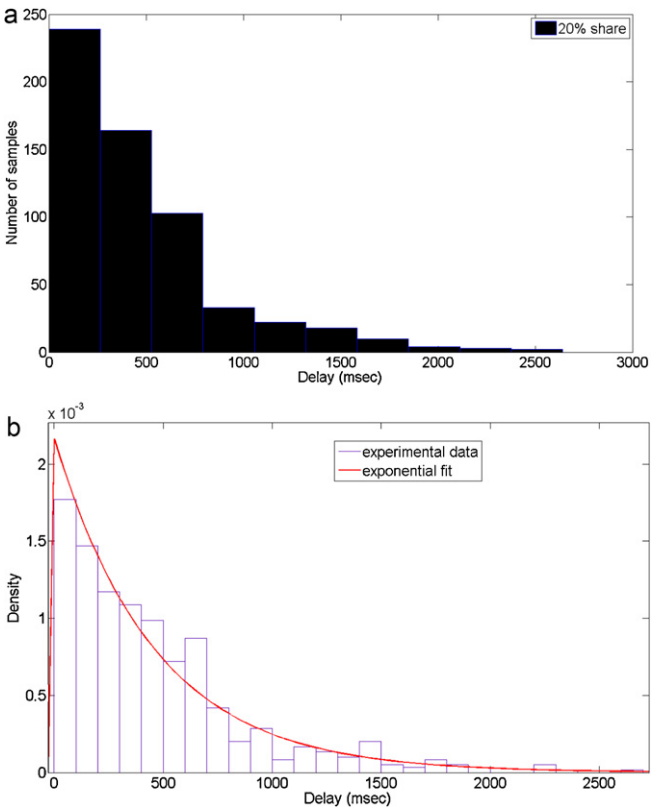


Fig. 13. (a) Distribution of the individual ping delays in 500 ms bins and (b) Density distribution of individual values (ping times) and fitting of an exponential distribution for scheduling period of 150 ms, pinging period of 0.5 s and CPU share of 60%. (a) Histogram of the distribution of the individual values and (b) density distribution and fitting of an exponential distribution.

6. Prediction of overhead

The experiments that were presented above are helpful for an infrastructure/Cloud provider in order to be able to have a generalized picture of how application performance deteriorates following the placement decisions. The high level conclusions could always be extracted as fuzzy logic rules in a similar decision making mechanism that would aid the provider during the scheduling

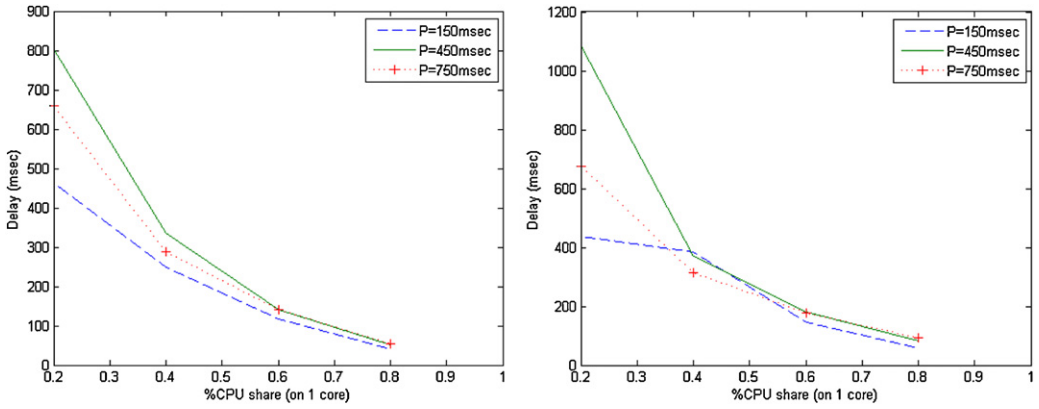


Fig. 12. (a) Mean time of the response times for full load and various configurations of granularity of assignment (scheduling period) *P* and pinging period 0.5 s and (b) Mdev time (as reported by the ping command) of the response times for full load and various configurations of granularity of assignment (scheduling period) *P* and pinging period 0.5 s.

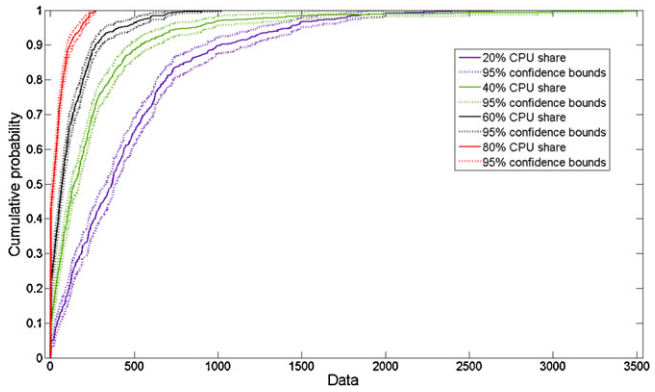


Fig. 14. Cumulative distribution function and according confidence intervals for all CPU shares, for scheduling period of 150 ms, pinging period of 0.5 s. The horizontal axis is the time delay in milliseconds and the vertical the probability to have a ping time less or equal to this delay.

and allocation policies enforcement. However, in order for this approach to meet its full potential, a suitable generic model should be designed. This model must be able to extract accurate predictions regarding the anticipated performance of the applications that have similar behavior to the investigated tests. It must be able to take as input the conditions of execution and produce the anticipated level of performance (test score), thus accurately quantifying the expected overhead.

In order to implement such a system, the most generic black-box method was chosen, an artificial neural network (ANN). ANNs are a methodology that aims to imitate the behavior of the human brain and have been used up to now for a number of applications (e.g. pattern recognition, function approximation, classification or time series prediction). Their main ability is to model a system's behavior based only on a training data set (black box approach) that includes a number of system inputs and the according outputs. In general, the ANN correlates the input with the output data in order to find an overall approximation of the model function that describes the dependence of the output from the input. It was chosen since as demonstrated in Section 5, there are numerous, even hidden, factors that may affect an application's performance like hardware design details. Having an analytical modeling approach is almost impossible, given the variety of these factors. But even if one devotes such an effort to create a detailed model, it will be deemed useless with the next processor generation or a different architectural design. The ANNs offer a much more generic and flexible approach, and given only a suitable data set, can achieve a quick and satisfactory model.

The identified inputs and outputs of the ANN model are depicted in Fig. 15 and include the investigated parameters of the previous sections.

The range or potential values of the parameters are listed in Table 4.

All the inputs and outputs were normalized in the (−1,1) numeric interval. For the non-numeric inputs, these were assigned

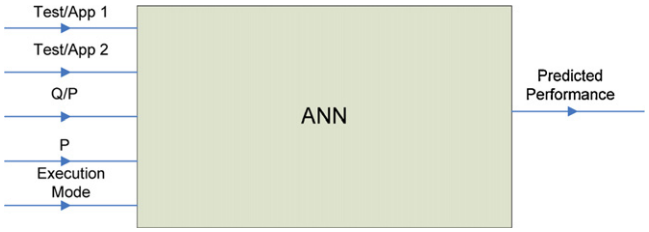


Fig. 15. ANN model.

Table 4
Range of parameters.

Test number/application type 1	Can be from 1 to 6 (Matlab benchmarks)
Test number/application type 1	Can be from 1 to 6 (Matlab benchmarks)
Q/*P* (core %)	0–80%
P	Arbitrary (in our case it is from 200 to 800 ms)
Execution mode	Can be standalone, same core, adjacent cores, non-adjacent cores

different values for each state. The normalization is necessary for internal ANN representation and better depiction of the system. The data set that was presented in the previous sections (1100 different executions, each for 500 s) was used for training (50%), intermediate validation during training (20%) and independent validation (30%) of the network.

6.1. Optimization of ANN structure and parameters

Despite their advantages, ANNs come with an important drawback. The decisions regarding their design parameters (like number of hidden layers, transfer functions for each layer and number of neurons per layer) are more or less based on the designer's instinct and experience. In order to automate and optimize this process, an evolutionary approach was followed in this paper. The aforementioned parameters of the ANN design were inserted in a Genetic algorithm (GA) (Holland, 1975), in order to optimize their selection. In each generation of the GA, a set of solutions is tried out, through training and validating the respective networks. The performance metric for each solution, in our case the error in the training set, is then returned to the GA, as an indication of its suitability. Then the next generation is created by the best solutions of the current one (elitism), combinations of existing solutions (crossover) and slight differentiation of them (mutation). Through this algorithm, the parameter space is quickly searched for a suboptimal solution. The parameters that are fed into the GA appear in Fig. 16.

In order to implement the aforementioned approach, an initial GA script is created in order to handle the population in each iteration and to launch the ANN creator script, after feeding it with the values that are depicted in the chromosomes. The ANN creator script takes the variables from the GA script and dynamically creates the ANNs according to the GA-selected parameters, by changing the network topology. Then it initializes and trains them (feed-forward, back propagation networks were used, trained with the Levenberg–Marquardt (Moré, 1978) algorithm) and returns the

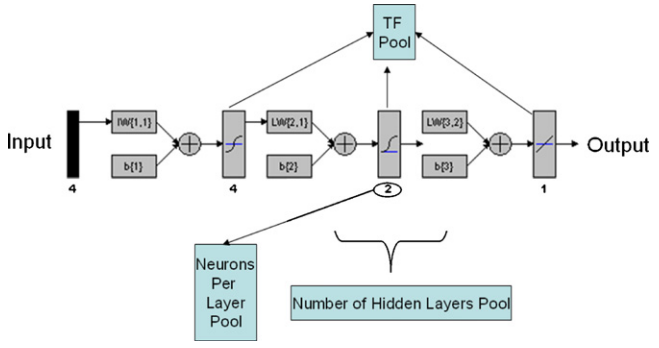


Fig. 16. Parameters of the ANN design taken from the GA. These include how many hidden layers will be initialized and what will their transfer function and their neurons. These parameters were in general decided by a human expert based on practical experience.

Table 5
Details of the best ANN structure as this was derived from the optimization process, for different number of generations of the GA.

Number of layers/GA generations	Neurons per layer	Transfer functions per layer	Mean absolute error (%)	Standard deviation	Time
4/30	5–7–19–1	Satlin–Radbas–Radbas–Satlins	5.94	16.47	40 min
3/50	5–7–1	Tansig–Tansig–Purelin	5.06	13.85	1.1 h
4/100	5–10–18–1	Radbas–Softmax–Radbas–Netinv	4.59	14.80	3 h
4/150	5–24–8–1	Purelin–Logsig–Logsig–Satlins	5.16	14.61	5 h

performance criterion (error in the training set) to the GA for the evaluation of each solution. An intermediate validation set was used in order to enhance the ANN generalization capabilities, the network's ability to predict cases that it has not met before during training. In this case, each network was trained by the 50% of the data set then if its predictions on the intermediate validation set had a mean absolute error (MAE) smaller than 10%, this network was saved. In the end of the generations, from all the saved networks, the final candidate was the one that showed the best performance (MAE) in the independent validation set. It was also investigated if the returned metric of the GA was the MAE in the intermediate validation set, instead of the training set error. This showed a worse performance.

13 different transfer functions were taken under consideration (as they are described in http://www.mathworks.com/help/toolbox/nnet/function.html) and a maximum limit of ten (10) layers and thirty (30) neurons per layer was inserted. Four independent runs were conducted for different number of generations (30, 50, 100 and 150). Each generation investigated 20 possible solutions.

The ANN structure and overall prediction accuracy for these is described in Table 5. In this table, only the best model from each run is depicted. The mean absolute error for all 328 independent validation cases is between 4.5 and 5.94, depending on the number of generations used, which can be considered as a very good estimator. Only the values in the independent validation set were used for the calculation of this error. These data have not been used during training.

The deterioration in the performance for 150 runs is not expected but can be attributed to the fact that each GA run depends on some random parameters, like the initial values of the chromosome or the randomness in the selection of characteristics to mutate. In a real-world situation, this can be easily tackled by adding generations of investigation until a specific performance criterion is met. The most suitable solutions are found in the 50 and 100 generations runs. The difference lies in the fact that for 100 generations a better mean error is acquired but a worse deviation metric. The final choice depends on the trade-off between these characteristics.

Another interesting conclusion is that smaller networks are more able to capture the dependencies between the input and the output. Even though the number of layers had a maximum value of 10, only 3 or 4-layer networks showed good performance (3 is the minimum limit for ANNs). This can be attributed to more robust behavior of small networks due to the following aspect. The influence of the *j*th input to the *i*th output depends on all the alternative paths that connect them and the increase of this path number (because of higher number of layers) seems to increase the standard deviation of the prediction. A similar result appears in the circuit synthesis domain, where the increase in the number of paths (ladder compared to lattice synthesis) results in higher system sensitivity.

In Fig. 17 the evolution of the mean absolute error of the saved networks is portrayed for each run. These networks are saved as the GA progresses and if the intermediate validation criterion is met (error in the intermediate validation set <10%). The lack of stability (continuously reduced mean error) in this process is due to

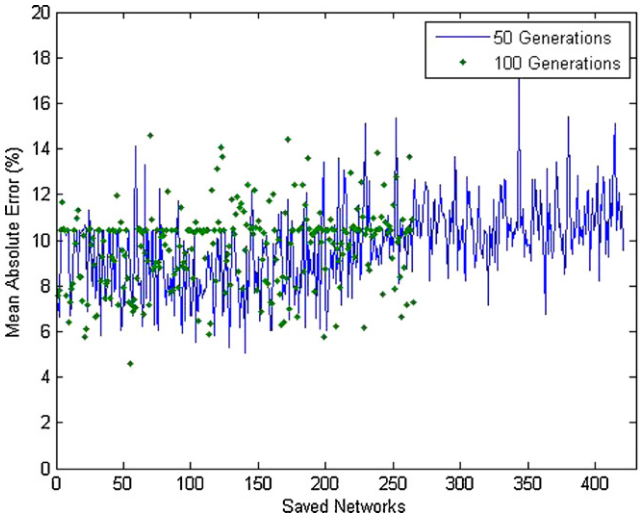


Fig. 17. Temporal evolution of the performance (mean absolute error on the independent validation set) of the networks that meet the save criterion (MAE < 10% on the intermediate validation set). After a period of time, the GA has converged to a suitable solution, after which the performance of the investigated networks deteriorates.

the fact that randomness is up to a point inherent in an evolutionary approach due to the use of operators such as mutation and crossover.

Furthermore, from this graph it is evident that a large number of good quality networks are produced through the proposed optimization process. It can also be seen that the networks that are saved in later generations of one run appear to be losing quality. This can be attributed to the fact that when the GA finds a good solution, it also focuses on nearby similar configurations. If the good solution is the local or global optimum, then the nearby solutions will be underperforming and a large number of the candidates will be devoted to this area in vain (other solutions based on crossover will search the remaining sample space). However this is also one of the strengths of GAs, since through this process in the early stages vast areas of the sample space are tried out and afterwards the best ones are thoroughly examined in order to find the optimal solution in that particular interval.

The distribution of the errors of the network in predicting the performance (test scores) is directly compared to the multi-variate regression method in Fig. 18. This accuracy is based only on the independent validation set only, for objectivity purposes.

6.2. Comparison with multi-variate linear regression

In order to compare our approach, the multivariate regression method was chosen, that is also used in Koh et al. (2007). The MATLAB function mvregress was used, in the same normalized data set. The only difference was that now the training and intermediate validation data vectors were merged. In this method, there is no need to have intermediate validation step. For the validation of the results, the same 30% independent validation data set was used, for objectivity purposes. The details of the approximation function

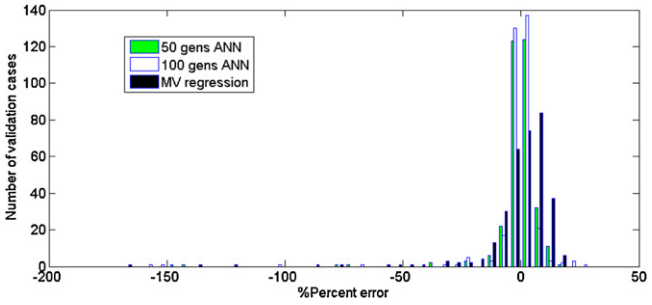


Fig. 18. Histogram of the errors in the independent validation set for 50 and 100 generation runs of the GA-ANN model compared to multi-variate regression. The difference lies on the center of the distributions, which for the proposed method is around 0 while for mv-regress is around 8%, The horizontal axis is the error (%) in the estimation (in bins) while the vertical is the number of validation cases for which the error was in the specific bin.

Table 6
Multi-variate regression details.

Coefficients	Mean absolute error	Standard deviation	Time needed
0.0218 −0.0171 0.8326 −0.0801 0.0631 0	8.78%	18.15	∼1 s

appear in Table 6. The coefficients cell contains in the respective order the multipliers for each one of the 5 inputs depicted in Fig. 15, plus the constant factor. Their values are low because they have been derived for the normalized values of the dataset in the [−1,1] interval.

The results from this method have been incorporated in Fig. 18 for direct comparison with the best ANN models from the optimization method.

From the comparison of the two approaches it can be concluded that the optimized ANNs have a better performance in terms of mean absolute error and standard deviation. This is also evident from the distribution of the errors as depicted in the histogram. While the ANN models exhibit high concentration of errors around the zero value, this is not the case for the regression model, which concentrates around the 8–10% error point.

The standard deviation values are high for both methods, but this is due to specific values of the 328-case validation set for which the measurements have large deviations. This happens for the peaks of the graphs presented in Section 5 and can be attributed to some random effects that affect system performance (e.g. system tasks that update the OS inside the VM, etc.), like in the case of $P = 250$ ms that was mentioned in the previous sections.

However, because the proposed method has much larger computational needs, for environments where new models must be created on the fly the multi-variate regression method offers a satisfactory performance in a very fast way. In the context of investigation in this paper (infrastructure/Cloud provider that wants to have a model for more efficient allocation of VMs over nodes), the computational needs of the optimized ANN method are not considered a constraint. The models will be created once, and may be refreshed in the future with new data. However, tight timing constraints for the production of the models do not exist. This is an off-line process.

Detailed percent improvements achieved by the compared method are depicted in Table 7.

For the response time of the models, that is the duration needed for each model to provide the prediction after the assertion of the conditions of execution to the input, this was better for the multivariate regression. However, for the ANN it was also very fast (∼10 ms), thus not making this a critical limitation.

Table 7
Comparison between multivariate regression and GA-optimized ANNs.

Method	%Improvement (ANN mean absolute error compared to mvregress)	%Improvement (ANN standard deviation compared to mvregress)
50 gen. GA-ANN	42.36%	23.64%
100 gen. GA-ANN	47.61%	18.4%

7. Conclusions and future work

As a conclusion, the emergence of virtualized and shared infrastructures like Clouds imposes a significant challenge for providers and applications. Applications that are running inside virtual machines are affected by many factors like virtualization and co-allocation of other VMs. In this paper, the effects of these allocation decisions were investigated, by taking under consideration a significant number of parameters like real time scheduling decisions, types of workload and different deployment scenarios. The performance overhead posed by these parameters can reach up to 150%, while carefully selecting the co-allocated tasks and the scenario of placement can minimize or even cancel this effect. What is more, higher scheduling periods result in a significant reduction in the overhead caused by the interference. Furthermore, the application of such a performance analysis method on new hardware design may lead to optimized multi-core architectures for Cloud-specific environments, through the identification of bottlenecks like the shared memory bus.

In order to provide an automatic decision making mechanism that will guide the infrastructures to proper configurations and remove the need of human intervention, a GA-optimized ANN can be used to model, quantify and accurately predict the performance of the applications for a given configuration. The model acquired from this process is effective (error < 5%), generic and can be at any time extended in order to include other identified significant factors or scenarios. Through the use of this mechanism, an infrastructure/Cloud provider can have a priori knowledge of the interference. Thus they are able to optimize the management of the physical resources.

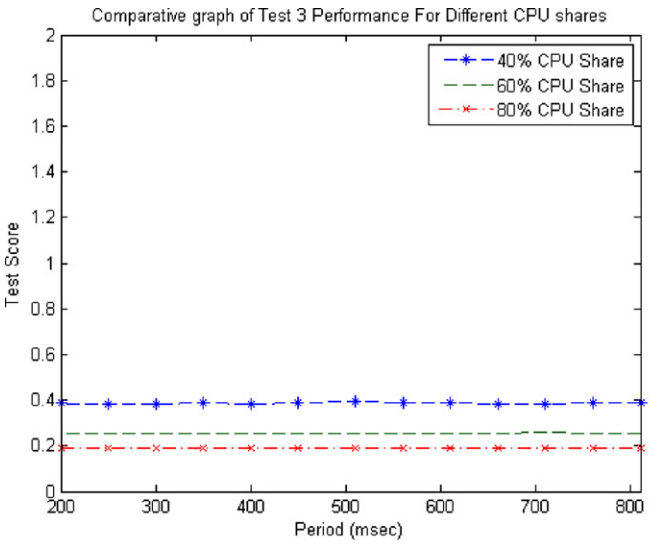
For the future, one interesting aspect to pursue is the automatic detection of the type of workload caused by each application. In this study, the analysis was based on the 6 Matlab benchmark tests, that depict different types of computations. For real-world applications this means that they must be matched to one or more of these elementary tests, based for example on the comparison of their footprints.

Acknowledgments

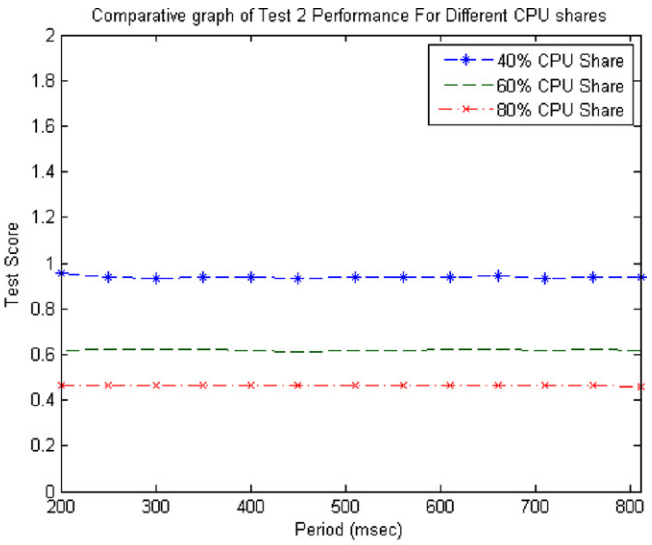
This research is partially funded by the European Commission as part of the European IST 7th Framework Program through the projects IRMOS under contract number 214777 and OPTIMIS under contract number 257115.

Appendix A. Remaining figures from Section 5.1

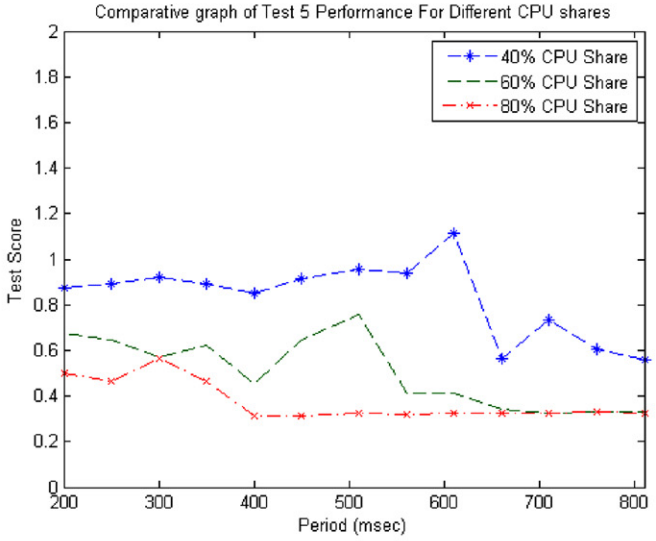
See Fig. A.1.



a) Standalone Test 3



b) Standalone Test 2



c) Standalone Test 5

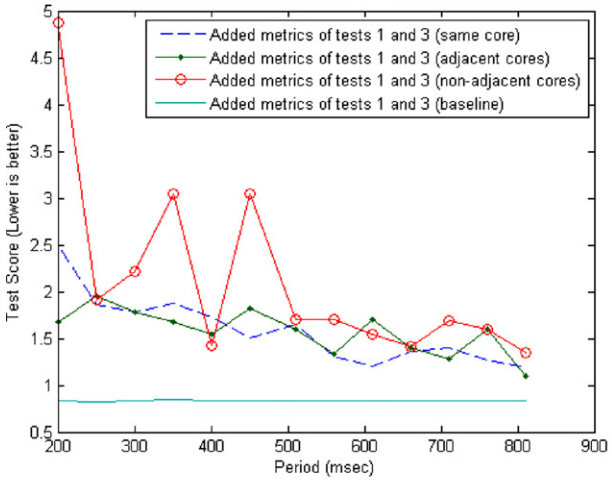
Fig. A.1. Remaining figures from Section 5.1 (Fig. 4: test scores for different CPU percentages and granularity (scheduling period) when the tests are running without any interference (co-scheduled tests). The test score (vertical axis) indicates the time needed for executing one test run and has been calculated from the mean value of hundreds of test runs inside a 500-s execution run. The scheduling period (horizontal axis) is the time interval in which the allocation of *X*% CPU share is guaranteed). (a) Standalone test 3; (b) standalone test 2; (c) standalone test 5.

Appendix B. Remaining combinations from Section 5.4

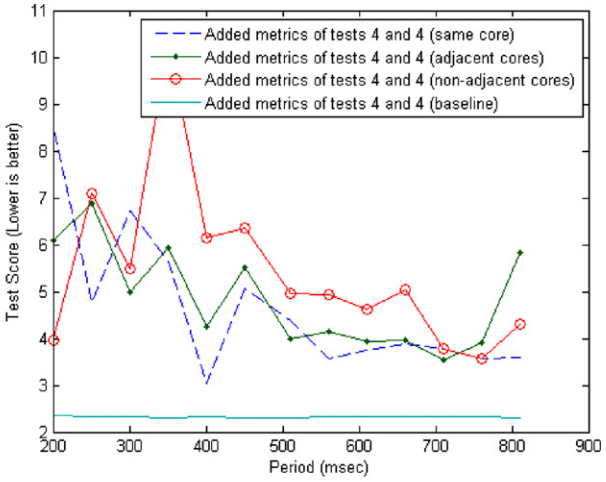
See Fig. B.1.

Appendix C. Figures for pinging period = 2 s (continuation of Section 5.5)

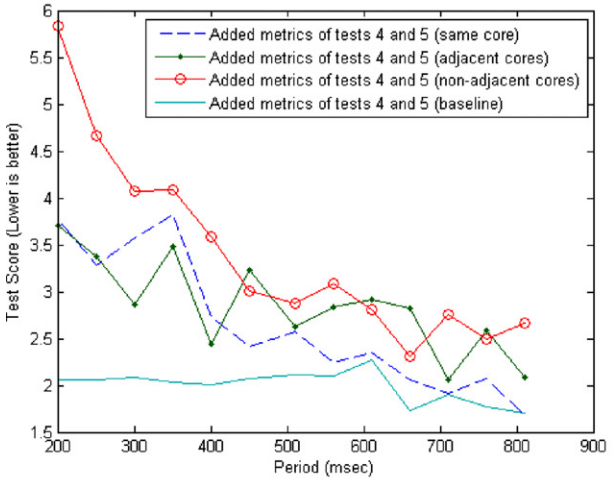
See Fig. C.1.



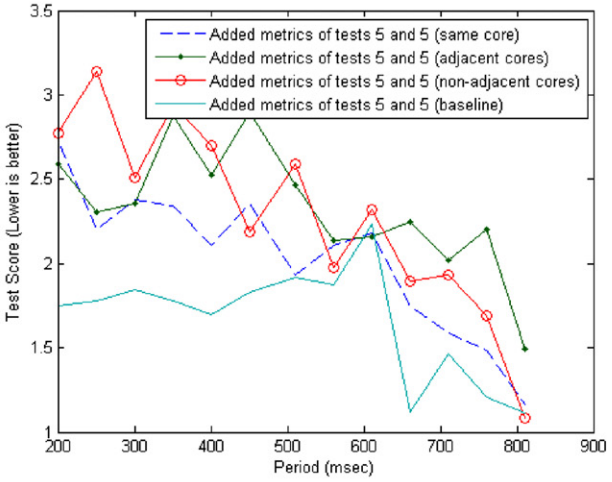
a) Tests 1 and 3 Combined Performance



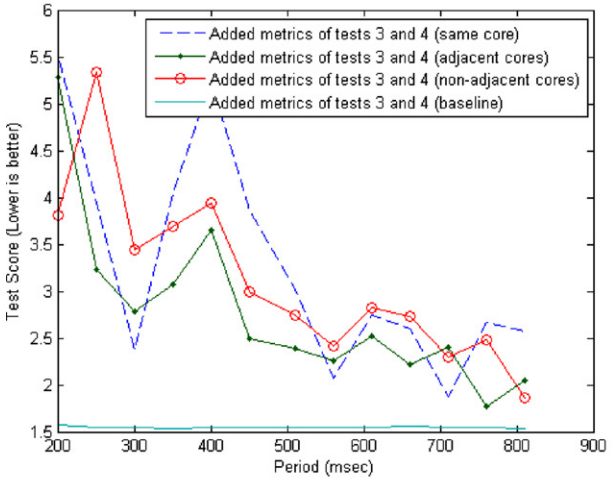
b) Tests 4 and 4 Combined Performance



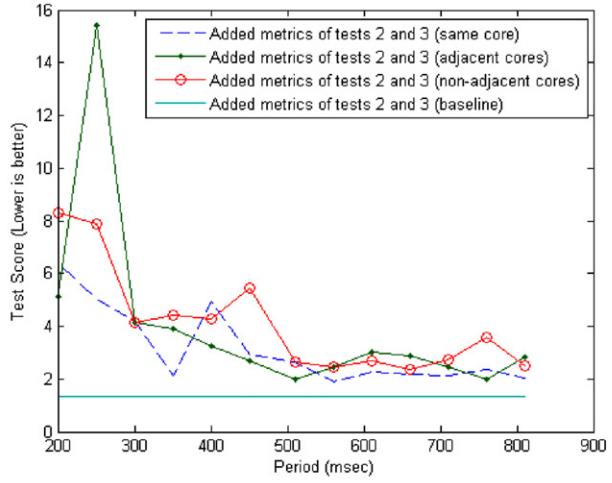
c) Tests 4 and 5 Combined Performance



d) Tests 5 and 5 Combined Performance

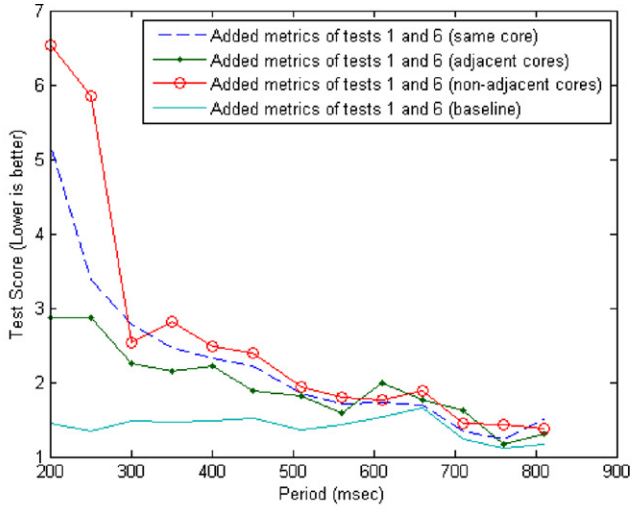


e) Tests 3 and 4 Combined Performance

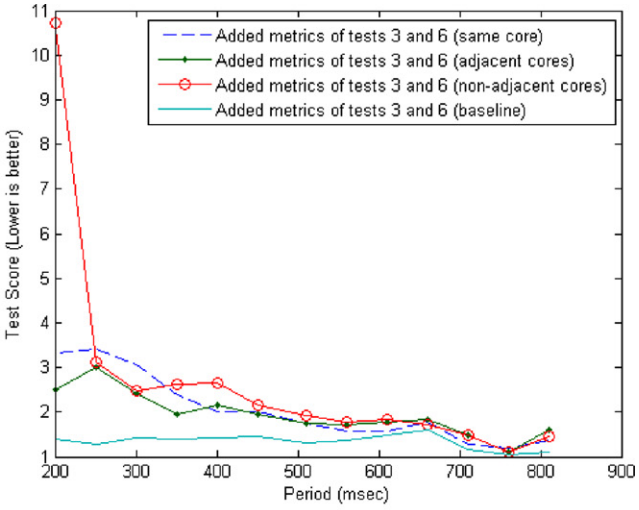


f) Tests 2 and 3 Combined Performance

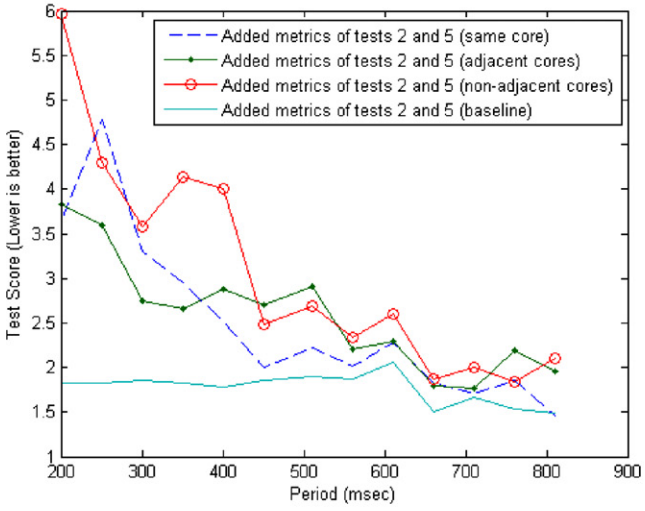
Fig. B.1. Remaining combinations from Section 5.4 (Fig. 6: comparison of system level (added scores) performance for all deployment scenarios and 40% CPU allocation per VM. In each of the graphs a specific combination of tests is investigated in order to compare their added scores (vertical axis) with regard to different deployment scenarios and how this performance is affected by the scheduling period *P*). (a) Tests 1 and 3 combined performance; (b) tests 4 and 4 combined performance; (c) tests 4 and 5 combined performance; (d) tests 5 and 5 combined performance; (e) tests 3 and 4 combined performance; (f) tests 2 and 3 combined performance; (g) tests 1 and 6 combined performance; (h) tests 3 and 6 combined performance; (i) tests 2 and 5 combined performance; (j) tests 2 and 2 combined performance; (k) tests 3 and 3 combined performance; (l) tests 1 and 2 combined performance.



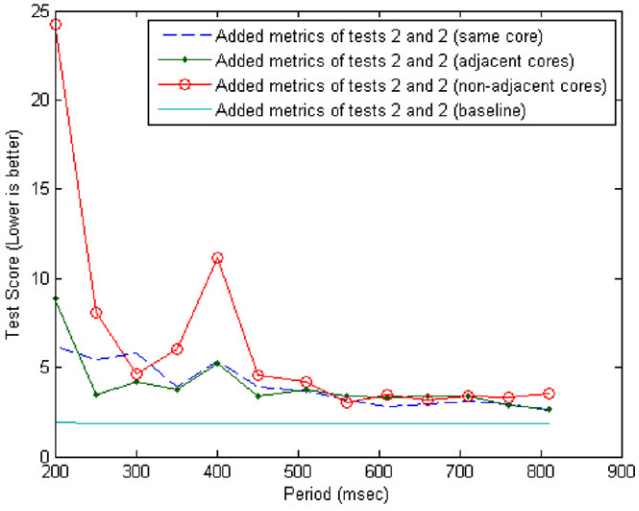
g) Tests 1 and 6 Combined Performance



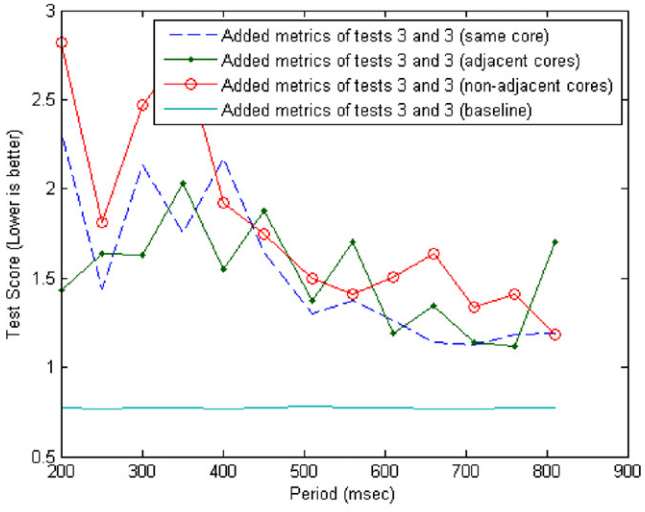
h) Tests 3 and 6 Combined Performance



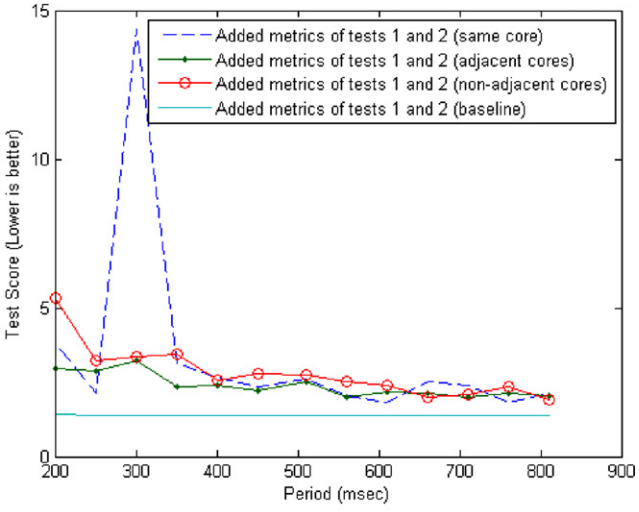
i) Tests 2 and 5 Combined Performance



j) Tests 2 and 2 Combined Performance



k) Tests 3 and 3 Combined Performance



l) Tests 1 and 2 Combined Performance

Fig. B.1. (*Continued*).

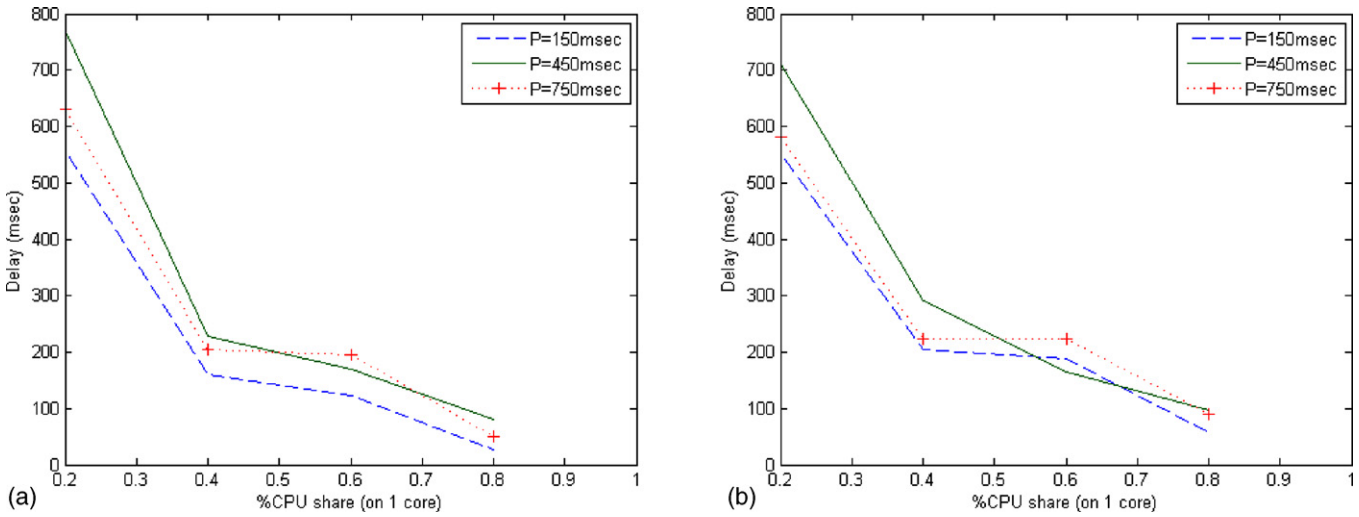
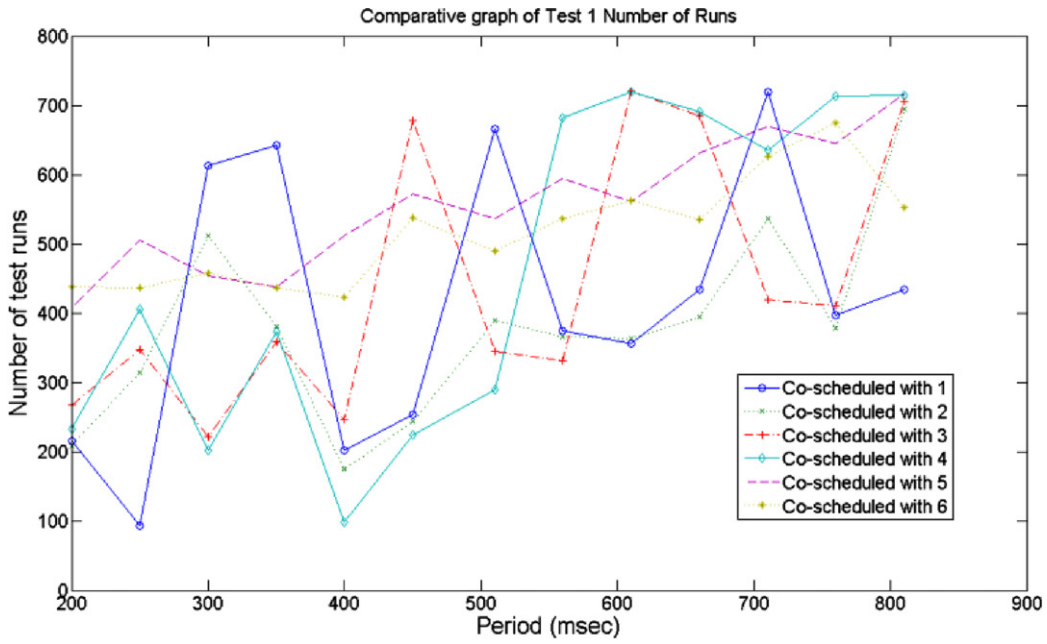


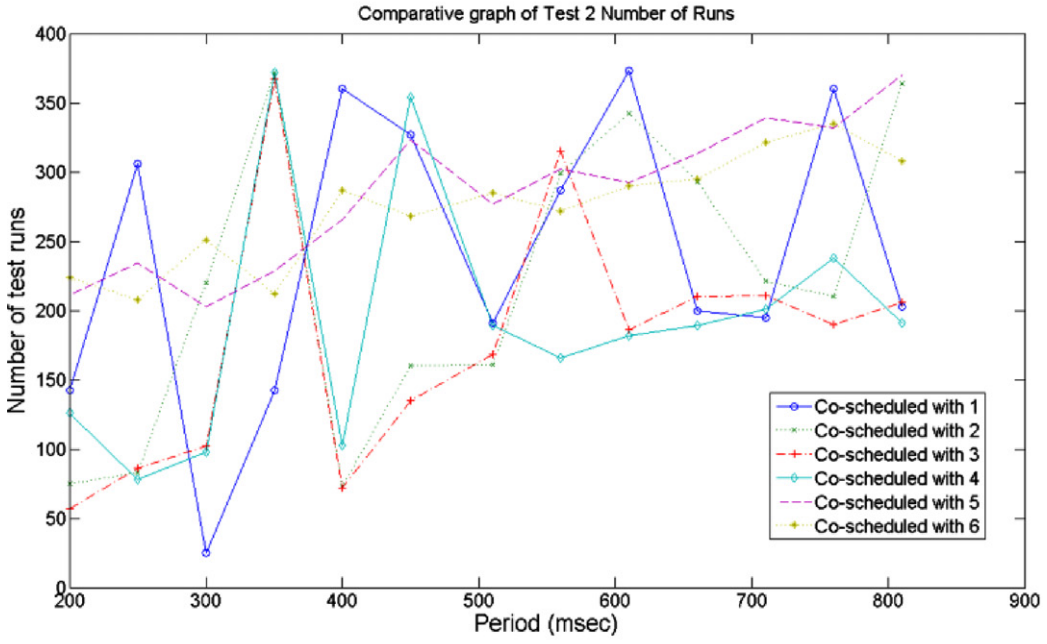
Fig. C.1. (a) Mean time of the response times for full load and various configurations of granularity of assignment (scheduling period) *P* and pinging period 2 s and (b) Mdev time (as reported by the ping command) of the response times for full load and various configurations of granularity of assignment (scheduling period) *P* and pinging period 2 s (continuation of Section 5.5).

Appendix D. Number of executions for each of the data points in same core scenario graphs

See Fig. D.1 .

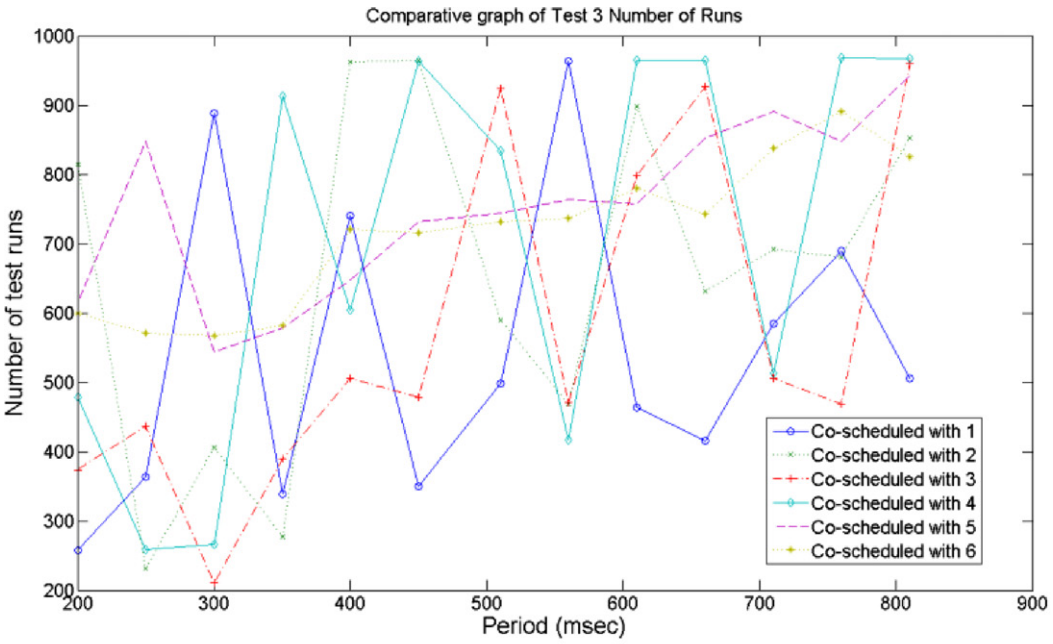


(a) Test 1 number of runs when co-scheduled with all other tests

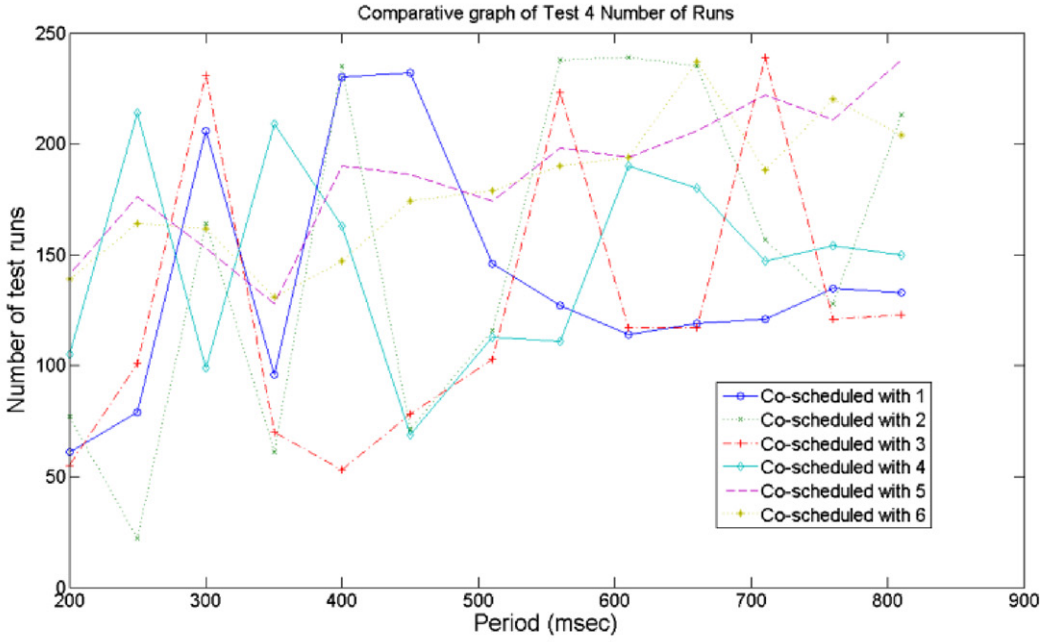


(b) Test 2 number of runs when co-scheduled with all other tests

Fig. D.1. Number of test runs from which the mean values for the scores were extracted for the same core scenario graphs (Fig. 5) for (a) test 1 when it is coscheduled with all other tests, (b) test 2 when it is coscheduled with all other tests, (c) test 3 when it is coscheduled with all other tests, (d) test 4 when it is coscheduled with all other tests, (e) test 5 when it is coscheduled with all other tests, (f) test 6 when it is coscheduled with all other tests. (a) Test 1 number of runs when co-scheduled with all other tests; (b) test 2 number of runs when co-scheduled with all other tests; (c) test 3 number of runs when co-scheduled with all other tests; (d) test 4 number of runs when co-scheduled with all other tests; (e) test 5 number of runs when co-scheduled with all other tests; (f) test 6 number of runs when co-scheduled with all other tests.

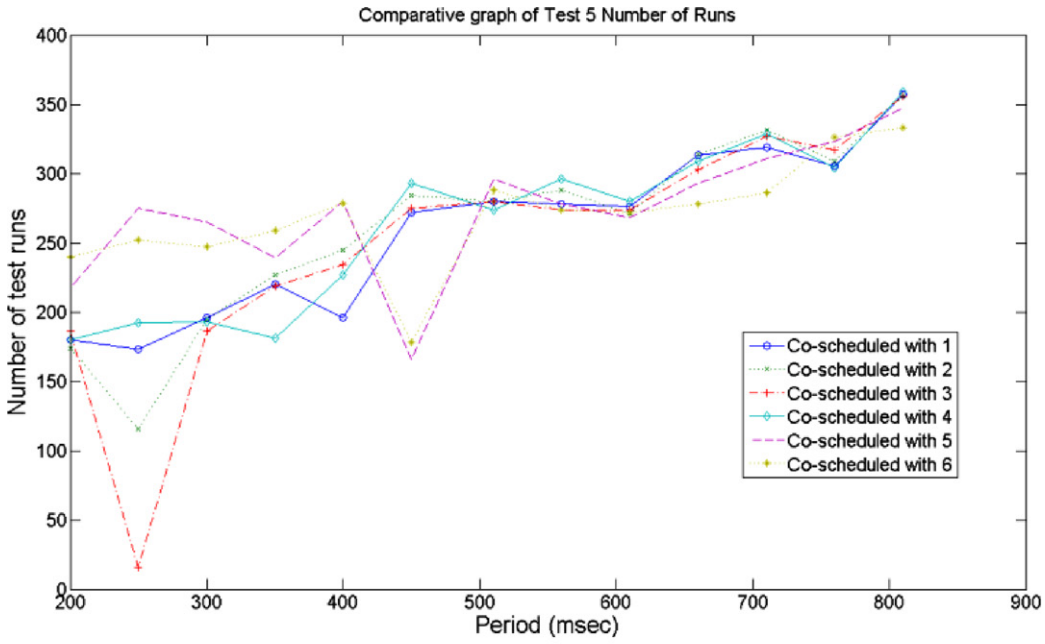


(c) Test 3 number of runs when co-scheduled with all other tests

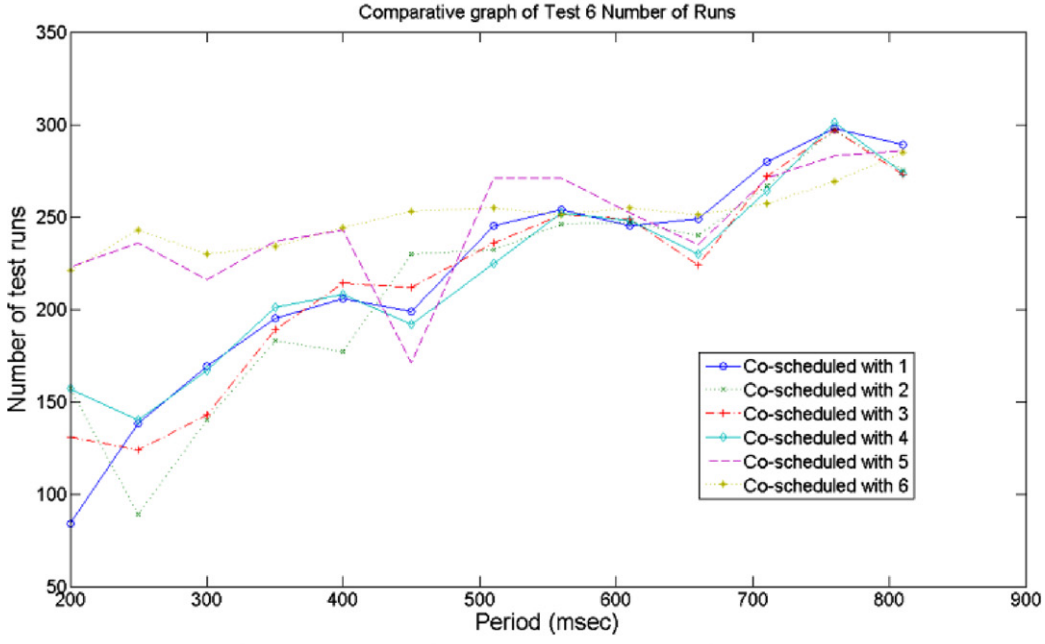


(d) Test 4 number of runs when co-scheduled with all other tests

Fig. D.1. (*Continued*).



(e) Test 5 number of runs when co-scheduled with all other tests



(f) Test 6 number of runs when co-scheduled with all other tests

Fig. D.1. (*Continued*).

References

Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M., 2010. A view of cloud computing. Commun. ACM 53 (4), 50–58, doi:10.1145/1721654.1721672, http://doi.acm.org/10.1145/1721654.1721672.

Checconi, F., Cucinotta, T., Faggioli, D., Lipari, G., 2009. Hierarchical multiprocessor CPU reservations for the Linux Kernel. In: Proceedings of the 5th International Workshop on Operating Systems Platforms for Embedded Real-Time Applications (OSPERT 2009) , Dublin, Ireland.

Cucinotta, T., Anastasi, G., Abeni, L., 2009. Respecting temporal constraints in virtualised services. In: Proceedings of the 2nd IEEE International Workshop on Real-Time Service-Oriented Architecture and Applications (RTSOAA 2009) , Seattle, Washington, July.

Cucinotta, T., Checconi, F., Kousiouris, G., Kyriazis, D., Varvarigou, T., Mazzetti, A., Zlatev, Z., Papay, J., Boniface, M., Berger, S., Lamp, D., Voith, T., Stein, M., 2010a. Virtualised e-Learning with real-time guarantees on the IRMOS platform. In: Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications (SOCA 2010) , Perth, Australia.

Cucinotta, T., Giani, D., Faggioli, D., Checconi, F., 2010b. Providing performance guarantees to virtual machines using real-time scheduling. In: Proceedings of the 5th Workshop on Virtualization and High-Performance Cloud Computing (VHPC 2010) , Ischia (Naples), Italy.

El-Refaey, M.A., Rizkaa, M.A., 2010. CloudGauge: a dynamic cloud and virtualization benchmarking suite. In: Enabling Technologies: Infrastructures for Collaborative Enterprises (WETICE), 2010 19th IEEE International Workshop on, vol., no., pp. 66–75, 28–30 June 2010, doi:10.1109/WETICE. 2010.17.

Fiszelew, A., Britos, P., Ochoa, A., Merlino, H., Fernández, E., García-Martínez, R., 2007. Finding optimal neural network architecture using genetic algorithms. Adv. Comput. Sci. Eng. Res. Comput. Sci. 27.
García-Guirado, A., Fernández-Pascual, R., García, J.M., 2009. Virtual-GEMS: An infrastructure to simulate virtual machines. In: Fifth Annual Workshop on Modeling, Benchmarking and Simulation MoBS 2009 held in Conjunction with the 36th Annual International Symposium on Computer Architecture Sunday.
Giustolisi, O., Simeone, V., 2006. Optimal design of artificial neural networks by a multi-objective strategy: groundwater level predictions/Construction optimale de réseaux de neurones artificiels selon une stratégie multi-objectifs: prévisions de niveau piézométrique. Hydrol. Sci. J. 51 (3), 502–523.
Holland, J., 1975. Adaptation in Natural and Artificial Systems. University of Michigan Press, Ann Arbor.
http://support.gams-software.com/doku.php?id=platform:aws.
http://view.eecs.berkeley.edu/wiki/Dwarfs.
http://www.irmosproject.eu/Postproduction.aspx.
http://www.javaworld.com/javaworld/jw-12-2000/jw-1229-traps.html?page=4.
http://www.linux-kvm.org/.
http://www.mathworks.com/help/techdoc/ref/bench.html.
http://www.mathworks.com/help/toolbox/nnet/function.html.
http://www.mathworks.com/matlabcentral/fileexchange/11984.
http://www.vmware.com/.
http://www.xen.org/.
Ilonen, J., Kamarainen, J.K., Lampinen, J., 2003. Differential evolution training algorithm for feed-forward neural networks. Neural Process. Lett. 17 (February (1)).
IRMOS Project D5.1.1, 2009. Models of real time applications on service oriented infrastructures, It-Innovation and other partners.
Jin, H., Cao, W., Yuan, P., Xie, X., 2008. VSCBenchmark: benchmark for dynamic server performance of virtualization technology. In: Proceedings of the 1st International Forum on Next-generation Multicore/Manycore Technologies , Cairo, Egypt.
Khalid, O., Maljevic, I., Anthony, R., Petridis, M., Parrott, K., Schulz, M., 2009. Dynamic scheduling of virtual machines running HPC workloads in scientific grids. In: New Technologies, Mobility and Security (NTMS), 2009 3rd International Conference on, vol., no., pp.1–5, 20–23 Dec., doi:10.1109/NTMS. 2009.5384725.
Koh, Y., Knauerhase, R.C., Brett, P., Bowman, M., Wen, Z., Pu, C., 2007. An analysis of performance interference effects in virtual environments. In: ISPASS ,. IEEE Computer Society, pp. 200–209.
Kousiouris, G., Checconi, F., Mazzetti, A., Zlatev, Z., Papay, J., Voith, T., Kyriazis, D., 2010. Distributed interactive real-time multimedia applications: a sampling and analysis framework. In: 1st International Workshop on Analysis Tools and Methodologies for Embedded and Real-time Systems (WATERS 2010) , Brussels, Belgium.
Leung, F.H.F., Lam, H.K., Ling, S.H., Tam, P.K.S., 2003. Tuning of the structure and parameters of a neural network using an improved genetic algorithm. IEEE Trans. Neural Networ. 14 (January (1)).
Liu, C.L., Layland, J., 1973. Scheduling algorithms for multiprogramming in a hard real-time environment. J. ACM 20 (1).
Makhija, V., et al., 2006. VMmark: a scalable benchmark for virtualized systems. Technical report, VMWare.
Moller, K.T., Virtual machine benchmarking. Diploma Thesis. Universitat Karlsruhe (TH). 2007.
Moré, J.J., 1978. The Levenberg–Marquardt algorithm: implementation and theory. In: Watson, G.A. (Ed.), Numerical Analysis, Dundee 1977. Lecture Notes in Mathematics, vol. 630. Springer, Berlin, pp. 105–116.
Padala, X.Z., Wanf, Z., Singhal, S., Shin, K., 2007. Performance evaluation of virtualization technologies for server consolidation. HP Labs, Tech. Rep. HPL-2007-59.
Soror, A.A., Minhas, U.F., Aboulnaga, A., Salem, K., Kokosielis, P., Kamath, S., 2008. Automatic virtual machine configuration for database workloads. ACM Trans. Database Syst. 35 (1), 47 p, Article 7, doi:10.1145/1670243.1670250, http://doi.acm.org/10.1145/1670243.1670250.
Tickoo, O., Iyer, R., Illikkal, R., Newell, D., 2010. Modeling virtual machine performance. ACM SIGMETRICS Perform. Eval. Rev. 37, 55.
Tikotekar, A., Vallïee, G., Naughton, T., Ong, H., Engelmann, C., Scott, S.L., Filippi, A.M., 2008. Effects of virtualization on a scientific application running a hyperspectral radiative transfer code on virtual machines. In: Proc. of HPCVirt'08. USA , ACM, pp. 16–23.
Weng, C., Wang, Z., Li, M., Lu, X., 2009. The hybrid scheduling framework for virtual machine systems VEE '09. In: Proceedings of the 2009 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments , ACM, pp. 111–120.
White, J., Pilbeam, A., 2010. A Survey of Virtualization Technologies With Performance Testing, ArXiv e-prints, 1010.3233.
Youseff, L., Butrico, M., Da Silva, D., 2008. Toward a unified ontology of cloud computing. In: Grid Computing Environments Workshop, 2008. GCE '08grid Computing Environments Workshop, 2008, GCE '08, pp. 1–10.

George Kousiouris received his diploma in Electrical and Computer Engineering from the University of Patras, Greece in 2005. He is currently pursuing his Ph.D. in Grid and Cloud computing at the Telecommunications Laboratory of the Dept. of Electrical and Computer Engineering of the National Technical University of Athens and is a researcher for the Institute of Communication and Computer Systems (ICCS). He has participated in the EU funded projects BEinGRID IRMOS and OPTIMIS and the National project GRID-APP. In the past he has worked for private telecommunications companies. His interests are mainly computational intelligence, optimization, computer networks and web services.

Tommaso Cucinotta graduated in Computer Engineering at the University of Pisa in 2000, and received the Ph.D. degree in Computer Engineering from the Scuola Superiore Sant'Anna of Pisa in 2004. He is Assistant Professor of Computer Engineering at the Real-Time Systems Laboratory (ReTiS) of Scuola Superiore Sant'Anna. His main research activities are in the areas of real-time and embedded systems, with a particular focus on real-time support for general-purpose Operating Systems, and security, with a particular focus on smart-card based authentication.

Theodora A. Varvarigou received the B. Tech degree from the National Technical University of Athens, Athens, Greece in 1988, the MS degrees in Electrical Engineering (1989) and in Computer Science (1991) from Stanford University, Stanford, California in 1989 and the Ph.D. degree from Stanford University as well in 1991. She worked at AT&T Bell Labs, Holmdel, New Jersey between 1991 and 1995. Between 1995 and 1997 she worked as an Assistant Professor at the Technical University of Crete, Chania, Greece. Since 1997 she was elected as an Assistant Professor while since 2007 she is a Professor at the National Technical University of Athens, and Director of the Postgraduate Course "Engineering Economics Systems". Prof. Varvarigou has great experience in the area of semantic web technologies, scheduling over distributed platforms, embedded systems and grid computing. In this area, she has published more than 150 papers in leading journals and conferences.