Profiling Cloud Applications with Hardware Performance Counters

Alexandre Kandalintsev and Renato Lo Cigno
University of Trento, Italy
{kandalintsev,locigno}@disi.unitn.it

Dzmitry Kliazovich and Pascal Bouvry
University of Luxembourg, Luxembourg
{dzmitry.kliazovich,pascal.bouvry}@uni.lu

***Abstract*—Virtualization is a key enabler technology for cloud computing. It allows applications to share computing, memory, storage, and network resources. However, physical resources are not standalone and the server infrastructure is not homogeneous. The CPU cores are commonly connected to the shared memory, caches, and computational units. As a result, the performance of cloud applications can be greatly affected if, while being executed at different computing cores, they compete for the same shared cache or network resource. The performance degradation can be as high as 50%. In this work we present a methodology which predicts the performance problems of cloud applications during their concurrent execution by looking at the hardware performance counters collected during their standalone execution. The proposed methodology fosters design of novel solutions for efficient resource allocation and scheduling.**

Keywords*—Cloud computing, virtualization performance, hardware performance counters

I. Introduction

Cloud computing becomes increasingly important. Its blend of flexible allocation and virtualization empowers scalability and reliability of applications, minimizing the burden posed on customers to fulfill these fundamental requirements. Virtualization decouples operating systems (OSs) and applications from hardware allowing easy migration (between different hardware and sites) and transparent upgrades.

The unleashed computing power enables new applications with virtually no bounds for scalability. At any time customers are assured that (almost) any demands for resources can be satisfied for an affordable price. So the core of a cloud is its management plane, which is responsible for reliability, scalability and efficiency.

Cloud computing efficiency has two main issues: *i)* how much of the equipment is needed to keep the cloud running and meet the Quality of Service (QoS) constraints negotiated in Service Level Agreements (SLA); *ii)* optimize operating expenses and minimize hardware power consumption. It is up to the management plane to decide either to "shrink" or "expand" the cloud. And this decision process is complex. It must account for a large number of parameters such as time, current load, load dynamics, size of the resource pool, networking, data center topology, and QoS [1]. The control plane enables/disables data center equipment and migrates VMs to different hardware as it is needed.

The virtualization layer ensures that VMs are logically isolated. However, even if VMs are perfectly isolated in the virtualized environment they still share hardware resources, and these resources are not infinite, i.e., if one VM uses them another VM has to wait. What is normally missing in cloud management planes is the consideration of mutual VMs' interference.

Most of the resource management algorithms consider CPU cores as unified resources adding performance in proportion to their number. Many consider a six-core CPU to be three times faster than a two-core CPU running at the same frequency, but this is an idealized situation. In reality the performance gain may vary due to the subsystems shared between cores. These subsystems can include CPU caches, memory bus, I/O lines, instruction decoders, branch predictors, computational units and other components. Therefore, under heavy load it is unlikely to see a linear gain in performance when adding more CPU cores. In fact, an increase the number of cores can even degrade VM performance for up to 50% due to the inter-VM interference [2], [3].

The level of interference varies and depends on many factors. As we discuss in this work, a proper placement of VMs can improve their performance significantly, while improper placement can become a cause of significant performance degradation. OS, libraries, programs and their versions, compilers used to build the system, hardware, BIOS settings are some of the factors that affect the performance results. It is difficult to predict how a particular VM will co-exist with other VMs in a given scenario. The most precise way to understand the loss in performance caused by a VM is to measure it. A good precision of this method is on the expense of its complexity – each VM should be executed with each other VM at least once during the measurement phase. It is possible for systems with limited number of executed VMs, but becomes impractical when the number of VMs is large.

Another factor contributing complexity is the VM diversity. There is a virtually infinite number of different virtual machines. But are they really so different in terms of interference? What if we find a simple way to rank arbitrary VMs according to the interference they cause to others? Such ranking may be imperfect, but it will certainly be useful for cloud management.

In this paper we present a novel methodology to classify and rank VMs based on the analysis of *Hardware Performance Counters* (HPCs)[1]. HPCs accumulate resource access statistics such as the number of time a VM accessed CPU caches or the success rate of the branch predictor. The main contributions of this paper are:

- The development of a methodology for profiling and

[1]In this work HPC always refers to hardware performance counters and should not be confused with high performance computing.

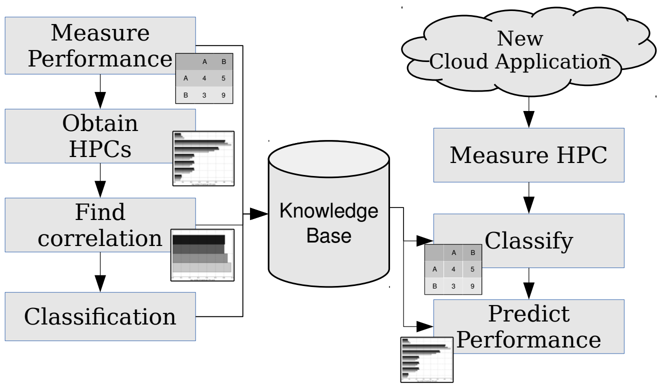

Fig. 1: The classification process at a glance.

ranking of VMs to estimate the level of their mutual interference;
- The evaluation of the methodology on x86-based and ARM-based hardware platforms;
- The analysis of the obtained profiling data to understand which shared resources are the main contributors to the VM interference.

II. Methodology

Fig. 1 presents a high-level overview of the classification process. It consists of two major phases: learning phase and working (classifying) phase. During the learning phase the system figures out which hardware counters are the most important for application profiling and how they are related to system performance. The obtained knowledge is stored in a database. The classification is performed by comparing the profile of a new task with classes from the database.

Each VM behaves differently in the presence of other VMs competing for computing, memory, or network resources: it can run unaffected, degrade or even in performance. If it is known which VMs co-exist well and which do not, we can perform correct placement and schedule their execution properly. The key question remains: *how to assess and predict VM interference?* A straightforward way is to launch all the VMs together in pairs and measure their mutual interference. However, this would take too much time. A better way is to profile the VMs individually and then, based on their profiles, reason how they will interact with each other.

Our goal is to classify VMs based on two parameters of interest – *sensitivity* and *interference* – and use them to guide resource allocation and scheduling. The sensitivity is a measure of how the performance of a given VM is affected by the activity of other VMs. On the contrary, the interference describes how the behavior of a given VM affects operation of neighboring VMs. As both the sensitivity and interference cannot be measured directly, we derive their values from the analysis of HPCs.

A. Hardware Performance Counters

HPCs are a mechanism for application profiling. HPCs are built-in CPU circuits designed to collect runtime low-level execution statistics. HPCs consist of two parts: event detectors and 64-bit registers (counters). Each time an event occurs the register associated with this kind of event is incremented.

HPC statistics include the frequency of access to instruction decoders, caches and Floating Point Unit (FPU).

B. Virtual Machines Profiling

The mapping between interference/sensitivity and the values of HPCs can be measured through correlation analysis. For this, we first calculate interference and sensitivity for a small set of VMs. This can be done by launching all pairs of the VMs and measuring their execution performances. Then we compute linear correlation by calculating *Pearson's product-moment correlation coefficient* (Pearson's r) between the interference/sensitivity and each of the hardware counters. The HPCs with strong correlation are then selected to predict interference/sensitivity values of an arbitrary VM. This prediction can be done by, for example, regression analysis.

III. Experimental Study

Our experiments are executed on a small scale heterogeneous testbed accounting for different architectures, using collection of different benchmark applications.

A. Testbed

We use the following equipment:

a) ARM Exynos: an "Odroid-U2" board based on Samsung Exynos-4412 system-on-chip with ARM Cortex-A9 four-core CPU clocked at 1.7GHz. ARM Exynos has 2 GB of RAM and 8 GB eMMC storage.

b) AMD FX: a board based on eight-core AMD FX-8120 CPU. The CPU consists of four two-core blocks, each equipped with its own 2 MB L2 cache. In addition, all two-core blocks share the same 8 MB L3 cache. In order to obtain stable and repeatable results the dynamic overclocking is disabled in BIOS. AMD FX is supplied with 16 GB DDR3-1600 RAM. A Crucial$^{\text{TM}}$M4 Solid State Drive (SSD) with 64 GB is used as a storage.

All measurements were done by "perf stat" command using all relevant counters reported by "perf list" command.

B. Benchmarks

Benchmarks are selected to provide a comprehensive comparison of cloud workloads. The emphasis is given to the real-world programs, although a few synthetic benchmarks (*matrix*, *blosc* and *integer*) are present as well.

1) ***blosc:*** a high performance compression library that optimizes data transfers between CPU and memory;
2) ***ffmpeg:*** transcoding a H264 FullHD video into 720p format;
3) ***integer:*** integer computations with the four operations;
4) ***matrix:*** a matrix multiplication benchmark based on gsl/blas library with the size of matrices of 2048*2048; data type is 64 bit float;
5) ***nginx:*** a web server benchmark focused on static files [4];
6) ***pgbench:*** a PostgreSQL database stress test [5];
7) ***sdag:*** a benchmark with machine learning [6];

8) ***sdagp:*** the same as sdag, but with a different memory layout;
9) ***wordpress:*** default installation of a popular blogging and publishing platform.

C. Software Architecture

Fig. 2 presents the software architecture of our experiments. The core component is the *Experiment Controller*. It sets-up VMs, checks OS settings, and launches the benchmarks.

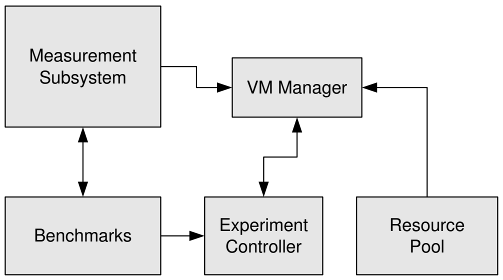

Fig. 2: Software architecture of the experiments

We implemented a specific *VM Manager*, a subsystem which provides virtualization method appropriate for the platform – QEMU for AMD FX and Linux Containers (LXC [7]) for ARM Exynos, as these platforms do not allow for standard VM management. The *Resource Pool* provides an abstraction layer for VMs to hardware resources. In our experiments each virtual machine was allocated 1 Gb of RAM and one core of the CPU. The *measurement subsystem* serves two different purposes. The first one is to collect the HPC statistics. The second purpose is to ensure that there is no activity in the system left unaccounted.

IV. Performance Results and Analysis

Tables I and II show how the VMs affect the performance of each other during their concurrent execution. Columns specify the names of the benchmarks currently being measured (*foreground* VMs), while rows are associated with the benchmarks executed at the same time on the neighboring core (*background*). The numeric values reported show the performance degradation of the foreground benchmarks with respect to their standalone execution. The dark grey cells correspond to the performance degradation of more than 15%, while the light grey cells show the degradation between 10% and 15%. The values reported in square boxes signal the performance increase. The latter can be achieved when the concurrent benchmark execution makes the use of the shared hardware resources (e.g., caches) more efficient than during standalone runs. For AMD FX we report interference results for both sibling cores that share the local cache and distant cores that share less resources.

These synoptic tables give several interesting insights. The first one is clear: running VMs on different cores does not ensure performance isolation. The degradation of performance is in some cases definitely high and can easily affect even the perceived QoS. Another interesting observation is that in the AMD FX architecture the interference is largely independent from the cores' distance. Finally, the performance improvement, which is at first sight counter-intuitive. First of all, the gain is usually small and in some cases it can well be just a measure noise, even if the measures are the average of many runs. Second, e.g., for caches, the algorithm that manages them is based on a very complex heuristic. Thus, the scenarios and setups in which the heuristic works better than others are not so surprising, specially taking into account that sharing resources is far more common than running in isolation, thus, the heuristic has been studied and tuned for these cases.

Table III presents the values of VM interference (how much the background affects the foreground) and sensitivity (how much a foreground is sensitive to have some other concurrent VM) calculated based on the performance degradation values reported in Tables I and II. The sensitivity is obtained as an average from the values each column, while the interference is an average on the rows. According to [8], the performance degradation increases following a power law with the number of CPU cores: 5% interference leads to $\sim 18.5\%$ and $\sim 33.6\%$ of overhead for four and eight cores respectively, but we cannot draw strict conclusions on this yet.

There are many more interesting measures that space forbids putting in this paper. The complete data set collected during experiments is available at [9].

A. Putting HPCs at work

Events can be different in nature, but all of them can be assigned a performance cost. For example, each LLC cache miss costs around 30-60 cycles of additional CPU time [10]. However, to understand the impact of the event on a system performance it is necessary to analyze the rate of the event occurrence in addition to the cost of the event. Low-frequency events do not contribute much to the VM interference. Therefore, we exclude low-frequency events from the analysis, even if they are costly. We operate with normalized frequency of events to avoid bias from the CPU clock rates.

Table III gives a high-level perception of the sensitivity and interference "properties" of the benchmarks. A quick investigation, to no surprise, indicates that all the top interfering benchmarks heavily use memory subsystem. Sdagp operates over a large set of scattered data. This requires a lot of memory access requests that cannot be served efficiently. Matrix is optimized for efficient memory access, but uses all available cache and constantly displaces other cached data. Blosc was designed to compress scientific data on-the-fly at extreme rates. It is capable of fully occupy the memory bus, which heavily impacts all other applications requesting bus access.

The high demand for memory resources that makes a benchmark an interferer, also makes it sensitive to the same resources. Therefore, there is a clear correlation between interference and sensitivity figures.

So far for pure empirical observations. Now we proceed to identify what HPCs are the most representative of the interference/sensitivity properties. We compute the correlation between each performance counter and the values of interference and sensitivity and focus further investigation on those counters with high correlation.

TABLE I: Performance degradation for concurrent execution of VMs running the benchmarks on ARM Exynos reported in percents

		Benchmark (foreground)								
		blosc	ffmpeg	integer	matrix	nginx	pgbench	sdag	sdagp	wordpress
Benchmark (background)	blosc	0.9	4.7	0.3	13.3	8.7	11.4	10.7	9.8	6.9
	ffmpeg	1.1	2.3	0.2	9.2	2.8	8.3	4.3	7.4	3.1
	integer	$\boxed{-0.7}$	0.0	$\boxed{-0.1}$	$\boxed{-0.1}$	$\boxed{-1.8}$	$\boxed{-0.8}$	0.1	0.7	$\boxed{-0.6}$
	matrix	9.6	8.4	0.3	15.4	21.9	24.7	22.8	41.3	14.0
	nginx	3.4	7.5	0.5	18.0	8.4	15.2	14.4	16.2	10.5
	pgbench	5.2	6.5	0.4	16.8	19.4	8.3	12.4	12.6	10.9
	sdag	0.1	2.6	0.2	8.5	4.9	10.8	5.3	9.0	3.7
	sdagp	4.2	10.0	0.3	15.4	20.1	27.8	24.0	50.3	14.8
	wordpress	3.1	4.8	0.2	9.8	9.3	15.5	11.1	12.9	6.8

TABLE II: Performance degradation for concurrent execution of VMs running the benchmarks on AMD FX reported in percents

(a) Sibling cores

		Benchmark (foreground)								
		blosc	ffmpeg	integer	matrix	nginx	pgbench	sdag	sdagp	wordpress
Benchmark (background)	blosc	5.2	0.8	0.4	8.4	3.3	7.9	1.5	4.3	2.1
	ffmpeg	3.7	$\boxed{-0.5}$	0.2	3.8	1.6	7.4	1.0	3.6	0.4
	integer	1.4	0.9	0.1	$\boxed{-0.7}$	0.5	5.8	$\boxed{-1.1}$	3.2	$\boxed{-0.8}$
	matrix	5.2	3.3	0.2	19.1	11.1	15.0	4.2	14.3	6.6
	nginx	4.3	1.1	0.4	12.9	5.8	11.9	3.0	12.3	3.5
	pgbench	2.6	1.6	0.1	5.7	2.6	9.3	$\boxed{-0.1}$	2.6	1.2
	sdag	1.1	$\boxed{-0.2}$	0.1	1.3	0.3	4.6	$\boxed{-1.2}$	$\boxed{-0.7}$	$\boxed{-0.5}$
	sdagp	2.8	$\boxed{-1.0}$	0.3	5.6	2.7	6.0	1.6	2.7	1.2
	wordpress	2.2	$\boxed{-0.1}$	$\boxed{-0.3}$	3.9	1.0	8.2	1.1	3.4	1.1

(b) Distant Cores

		Benchmark (foreground)								
		blosc	ffmpeg	integer	matrix	nginx	pgbench	sdag	sdagp	wordpress
Benchmark (background)	blosc	$\boxed{3.5}$	1.4	0.2	7.7	3.3	12.5	2.1	4.7	1.5
	ffmpeg	2.3	0.4	0.4	4.6	1.0	7.7	0.9	3.5	0.2
	integer	2.3	0.7	0.0	0.0	0.3	10.0	$\boxed{-0.6}$	3.5	$\boxed{-0.5}$
	matrix	5.2	3.8	1.1	19.0	11.7	15.1	4.9	17.6	6.1
	nginx	4.5	1.7	0.1	7.5	4.3	12.1	2.5	12.7	3.1
	pgbench	2.9	0.6	0.4	4.8	2.9	9.2	2.3	10.4	1.2
	sdag	1.4	1.1	0.0	2.1	0.5	10.5	$\boxed{-0.4}$	9.2	0.0
	sdagp	2.5	$\boxed{-0.5}$	0.3	4.2	1.9	8.9	1.9	3.8	1.1
	wordpress	3.3	0.2	0.5	4.8	1.1	12.1	0.4	11.5	0.4

TABLE III: Interference and sensitivity of benchmarks

(a) ARM Exynos

Interference		Sensitivity	
sdagp	18.6%	sdagp	17.8%
matrix	17.6%	pgbench	13.5%
nginx	10.4%	matrix	11.8%
pgbench	10.3%	sdag	11.7%
wordpress	8.2%	nginx	10.4%
blosc	7.4%	wordpress	7.8%
sdag	5.0%	ffmpeg	5.2%
ffmpeg	4.3%	blosc	3.0%
integer	-0.4%	integer	0.3%

(b) AMD FX: Sibling Cores

Interference		Sensitivity	
matrix	8.8%	pgbench	8.4%
nginx	6.1%	matrix	6.7%
blosc	3.8%	sdagp	5.1%
pgbench	2.8%	nginx	3.2%
sdagp	2.4%	blosc	3.1%
ffmpeg	2.4%	wordpress	1.7%
wordpress	2.3%	sdag	1.1%
integer	1.0%	ffmpeg	0.6%
sdag	0.5%	integer	0.2%

(c) AMD FX: Distant Cores

Interference		Sensitivity	
matrix	9.4%	pgbench	10.9%
nginx	5.4%	sdagp	8.6%
blosc	4.1%	matrix	6.1%
pgbench	3.9%	blosc	3.1%
wordpress	3.8%	nginx	3.0%
sdag	2.7%	sdag	1.6%
sdagp	2.7%	wordpress	1.5%
ffmpeg	2.3%	ffmpeg	1.0%
integer	1.7%	integer	0.3%

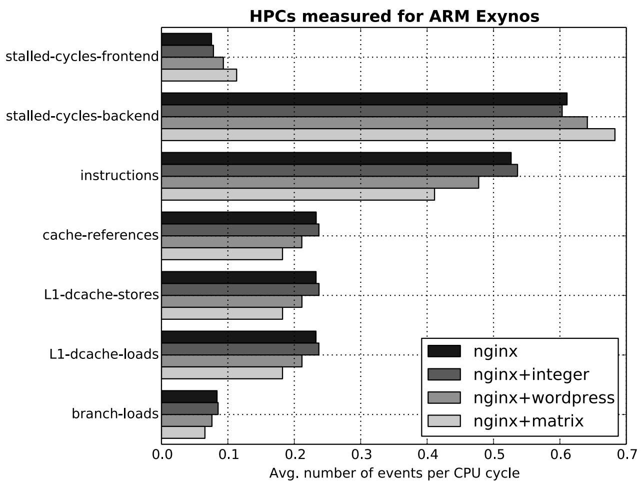

Fig. 3: Four cases of interference for ARM Exynos: no interference (only *nginx* is running), negative interference (*nginx* runs with *integer*), medium interference (*nginx* with *wordpress*) and strong (*nginx* with *matrix*).

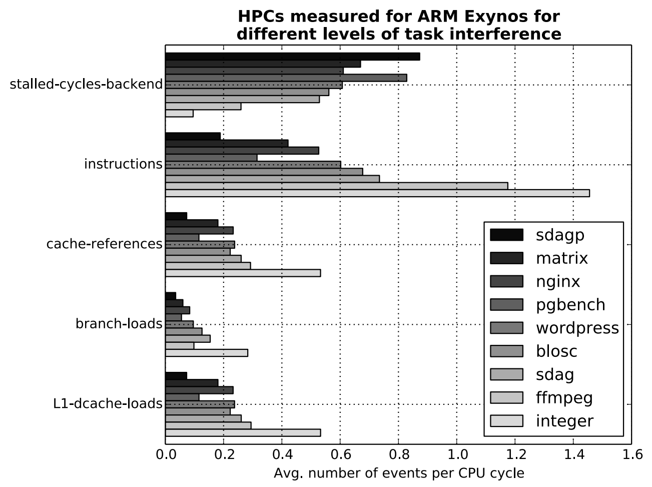

Fig. 4: Execution profiles of benchmarks running on ARM Exynos. Benchmarks are arranged according to their *interference* factors.

Fig. 3 presents HPC measurements for different levels of task interference: no interference (nginx alone), low interference (nginx+integer), medium interference (nginx+wordpress), and high interference (nginx+matrix). Fig. 4 shows the measured HPCs counters for all the benchmarks on the Exynos, corresponding to the interference values of Table IIIa. As expected, for low interference there are no significant changes in HPCs values. This means benchmarks execute as if they were alone. However, when interference becomes significant the HPCs values reflect task competition for the resources.

Surprisingly, there is no increase in any memory-related counters, which indicates that the main memory is not a primary bottleneck: the bottleneck arise inside the CPU before the main memory is accessed. The CPU cannot dedicate enough internal resources for all active cores. The cores compete for the resources, and this race creates a lot of pipeline stalls. This is reflected by "stalled-cycles-backend" parameter.

Referring again to Table II we now interpret results based on the HPC analysis. For sibling cores (Table IIa), there are many cases of performance improvement (represented with negative values of interference). This is especially evident for ffmpeg benchmark. The reason for performance improvements becomes evident from the analysis of two HPC counters: TLB and L1 caches. These parameters indicate that some data (probably kernel code) is shared between VMs. Shared data may speedup the simultaneous execution because if one core accesses it there is a chance that another core already fetched it and stored in shared cache. Another possible reason is that the overhead for keeping cache lines coherent is lower for the processes running on sibling cores [2]. The average per-task interference is around 3%.

For distant cores (Table IIb) the average per-task interference is equal to 4%. It is higher than for the sibling cores which is due to the fact that sibling cores can share data more efficiently. The picture is quite similar to the case with sibling cores. There is no single largest contributing counter to the interference. The average per-VM interference is 4%. This is slightly higher than the previous case and might be due to cache coherency protocol.

TABLE V: Hardware efficiency[1].

	IPC			Performance		
Bench	ARM	AMD	Ratio	ARM	AMD	Ratio (Norm.)
blosc	0.68	1.10	1.6	49.53s	12.7s	3.9 (2.1)
ffmpeg	1.18	1.36	1.2	689s	48.2s	14.3 (7.8)
integer	1.46	0.57	0.4	16.8s	15.3s	1.1 (0.6)
matrix	0.42	1.16	2.7	84s	16.8s	5.0 (2.8)
nginx	0.52	0.77	1.5	525MB/s	631MB/s	1.2 (0.7)
pgbench	0.31	0.52	1.7	155 tr/s	1293 tr/s	8.3 (4.6)
sdag	0.73	0.99	1.4	24.9s	8.3s	3.0 (1.7)
sdagp	0.19	0.44	2.3	132s	30s	4.4 (2.4)
wordpress	0.60	0.82	1.4	8.45r/s	7.19r/s	0.85 (0.5)

[1] Values in parentheses shows performance ratios normalized to CPU frequencies.

Table V presents efficiency of the hardware platforms in terms of the Instructions Per (CPU) Cycle (IPC). The ARM Exynos has substantially smaller IPCs. Interestingly, higher number of IPC does not necessarily lead to a higher performance per MHz. This is due to the differences in hardware architectures and optimization of compilers.

The ARM platform has the following performance issues for ffmpeg and pgbench benchmarks. The ffmpeg benchmark is not optimized for this platform [11]. The results for pgbench can be limited by weak storage subsystem.

The experimental results presented in this section unveil clear differences between the analyzed hardware platforms. In general, ARM cores have less optimization features than traditional x86 CPUs. They do "less job" per CPU cycle. For both platforms the most interfering tasks are the tasks that do heavy memory use (matrix, nginx, sdagp and blosc). This proves that the memory-related subsystems are the biggest bottleneck of general-purpose CPUs [12]. This bottleneck makes them sensitive as well because their performance almost entirely depends on data availability.

We can conclude that ARM Exynos performs well in-

TABLE IV: Correlation between interference, sensitivity and HPC

(a) ARM Exynos

Interference		
Parameter	Correlation	P-value
stalled-cycles-backend	0.887	0.1%
cache-misses	0.712	3.1%
L1-dcache-store-misses	0.712	3.2%
L1-dcache-load-misses	0.711	3.2%
L1-dcache-stores	-0.808	0.8%
branch-loads	-0.810	0.8%
instructions	-0.851	0.4%
Sensitivity		
Parameter	*Correlation*	P-value
stalled-cycles-backend	0.804	0.9%
branch-loads	-0.769	1.5%
cache-references	-0.830	0.6%
L1-dcache-loads	-0.831	0.6%
branch-instructions	-0.832	0.5%
instructions	-0.851	0.4%

(b) AMD FX

	Parameter	Correlation	P-value
Sibling cores	*Interference*		
	LLC-stores	0.915	0.1%
	L1-dcache-stores	0.732	2.5%
	Sensitivity		
	stalled-cycles-frontend	0.753	1.9%
	LLC-stores	0.694	3.8%
	cycles	-0.743	2.2%
Distant cores	*Interference*		
	LLC-stores	0.881	0.2%
	L1-dcache-stores	0.736	2.4%
	L1-dcache-prefetches	0.720	2.9%
	stalled-cycles-backend	-0.694	3.8%
	Sensitivity		
	L1-dcache-prefetch-misses	0.691	3.9%
	iTLB-load-misses	0.683	4.3%
	cycles	-0.775	1.4%

teger operations and web-servicing stuff (wordpress and nginx benchmarks). Heavy memory-intensive applications (sdag/sdagp, matrix and blosc benchmarks) perform better on AMD FX.

B. Lessons Learned

During the experiments we faced a number of technical problems. In the following we list the most relevant of them.

1) The HPC implementations vary across platforms. Not only the number of available events differs across platforms, but also their meaning. We checked OS Linux sources and developer manuals to ensure that our interpretation is correct.
2) We observed that VMs may shortly migrate to another CPU even if they are "pinned" to specific CPU cores. These cases are rare and do not change the overall picture.
3) Care should be taken when a large number of events is enabled. The number of available events exceeds the number of counting registers by a factor of 5 to 10. If too many events are enabled simultaneously, then the operating system has to do time multiplexing which leads to loss of precision.
4) Drivers and I/O can significantly affect the performance. We observed up to 40% deviation in instructions per second on heavy benchmarks if the system flushes disk caches. This does not affect the long-term average performance, but becomes critical for periodic measurements.

V. Summary and Future Work

In this paper we proposed a novel methodology for predicting performance of concurrently executed cloud applications by the analysis of hardware performance counters during their standalone execution. This becomes especially useful during resources allocation and scheduling in the large-scale computing systems that process incoming requests on-demand.

Future work will be focused on the evaluation of the proposed methodology on a larger number of hardware platforms, exploring trace points as an alternative profiling method of applications, and developing a CPU scheduler based on the designed methodology.

VI. Acknowledgment

The authors would like to acknowledge the funding from National Research Fund, Luxembourg in the framework of ECO-CLOUD project (C12/IS/3977641).

References

[1] M. Guzek, D. Kliazovich, and P. Bouvry, "A holistic model for resource representation in virtualized cloud computing data centers," in *IEEE 5th Int. Conf. on Cloud Computing Technology and Science (CloudCom)*, 2013.

[2] Q. Zhao, D. Koh, S. Raza, D. Bruening, W.-F. Wong, and S. Amarasinghe, "Dynamic cache contention detection in multi-threaded applications," in *7th ACM SIGPLAN/SIGOPS Int. conf. on virtual execution environments*, vol. 46, no. 7, 2011, pp. 27–38.

[3] M. A. Al-Mouhamed and K. A. Daud, "Experimental analysis of smp scalability in the presence of coherence traffic and snoop filtering," in *IEEE 14th Int. Conf. on High Performance Computing and Communication & 9th Int. Conf. on Embedded Software and Systems (HPCC-ICESS)*, 2012, pp. 81–88.

[4] "Nginx web server." [Online]. Available: http://nginx.org/en/

[5] *PostgreSQL Manual*, 2013. [Online]. Available: http://www.postgresql.org/docs/devel/static/pgbench.html

[6] A. Severyn and A. Moschitti, "Fast support vector machines for structural kernels," in *Proc. of ECML/PKDD*, 5-9 September, 2011, pp. 175–190.

[7] "Linux containers – chroot on steroids." [Online]. Available: http://lxc.sourceforge.net/

[8] A. Kandalintsev and R. Lo Cigno, "A behavioral first order cpu performance model for clouds' management," in *Proc. of 4th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*. IEEE, 2012, pp. 40–48.

[9] "Dataset, measures and analysis for profiling cloud applications." [Online]. Available: http://disi.unitn.it/locigno/profiling_clouds/

[10] R. Sheikh and M. Kharbutli, "Improving cache performance by combining cost-sensitivity and locality principles in cache replacement algorithms," in *Proc. of IEEE International Conference on Computer Design (ICCD)*. IEEE, 2010, pp. 76–83.

[11] Z. Ou, B. Pang, Y. Deng, J. K. Nurminen, A. Ylä-Jääski, and P. Hui, "Energy- and cost-efficiency analysis of arm-based clusters," in *CCGRID*, 2012, pp. 115–123.

[12] M. Ferdman, A. Adileh, O. Kocberber, S. Volos, M. Alisafaee, D. Jevdjic, C. Kaynak, A. D. Popescu, A. Ailamaki, and B. Falsafi, "Clearing the clouds: a study of emerging scale-out workloads on modern hardware," in *ACM SIGARCH Computer Architecture News*, vol. 40, no. 1. ACM, 2012, pp. 37–48.