View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Open Archive Toulouse Archive Ouverte

Open Archive Toulouse Archive Ouverte

OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible

This is an author's version published in:
http://oatao.univ-toulouse.fr/22300

Official URL
https://doi.org/10.1007/978-3-319-64203-1_24

To cite this version: Djomgwe Teabe, Boris and Wapet, Patrick Lavoisier and Tchana, Alain and Hagimont, Daniel *Dealing with Performance Unpredictability in an Asymmetric Multicore Processor Cloud.* (2017) In: European Conference on Parallel Processing (Euro-Par 2017), 28 August 2017 - 1 September 2017 (Santiago de Compostela, Spain).

Any correspondence concerning this service should be sent to the repository administrator: tech-oatao@listes-diff.inp-toulouse.fr

Dealing with Performance Unpredictability in an Asymmetric Multicore Processor Cloud

Boris Teabe(✉), Patrick Lavoisier Wapet, Alain Tchana, and Daniel Hagimont

University of Toulouse, Toulouse, France
{boris.teabedjomgwe,patrick.wapet,alain.tchana,
daniel.hagimon}@enseeiht.fr

Abstract. In a Cloud computing data center and especially in a IaaS (Infrastructure as a Service), performance predictability is one of the most important challenges. For a given allocated virtual machine (VM) in one IaaS, a client expects his application to perform identically whatever is the hosting physical server or its resource management strategy. However, performance predictability is very difficult to enforce in a heterogeneous hardware environment where machines do not have identical performance characteristics, and even more difficult when machines are internally heterogeneous as for Asymmetric Multicore Processor machines. In this paper, we introduce a VM scheduler extension which takes into account hardware performance heterogeneity of Asymmetric Multicore Processor machines in the cloud. Based on our analysis of the problem, we designed and implemented two solutions: the first weights CPU allocations according to core performance, while the second adapts CPU allocations to reach a given instruction execution rate (Ips) regardless the core types. We demonstrate that such scheduler extensions can enforce predictability with a negligible overhead on application performance.

1 Introduction

Cloud Computing enables remote access to on-demand allocated resources. The most popular cloud model is the so-called Infrastructure as a Service (IaaS) model, since it offers a high flexibility to cloud users. In order to provide isolation, IaaS clouds are often virtualized so that resources are allocated in terms of virtual machines (VMs). The provider defines a VM catalog (e.g. t2.medium, t2.small in Amazon EC2) presenting VM configurations which can be requested by cloud users. A VM configuration defines a capacity for each resource types, that we call virtual resource types as machines are virtual. Capacities are expressed as follows:

- the capacity of the network is expressed in terms of a bandwidth value (e.g. 100 MBps).
- the capacity of the hard disk is expressed in terms of both an IO bandwidth (e.g. 100 MBps) and a storage space (e.g. 1 TB).
- the capacity of the RAM is expressed in terms of a storage space (e.g. 10 GB).

DOI: 10.1007/978-3-319-64203-1_24

– the capacity of the CPU is expressed in terms of a number of virtual CPU (noted vCPU, e.g. 4 vCPUs).

The analysis of the above capacity expressions raises one question: **from the user point of view, what is the real capacity of each virtual resource type, given that virtual resources are mapped to heterogeneous physical resources**? Concerning both the network and the hard disk, the answer is quite clear because they are expressed using absolute units (independent from the underlying hardware). This is not the case for the two other virtual resources. Both the RAM bandwidth (which is not presented to the user) and the vCPU computing capacity depend on the underlying hardware. Figure 1 left shows that the same VM type from Rackspace and Azure cloud delivers different performance levels according to the underlying processor. This results in the problem of performance unpredictability [16], which has been identified by Microsoft [2] as part of the five top significant challenges in the cloud.

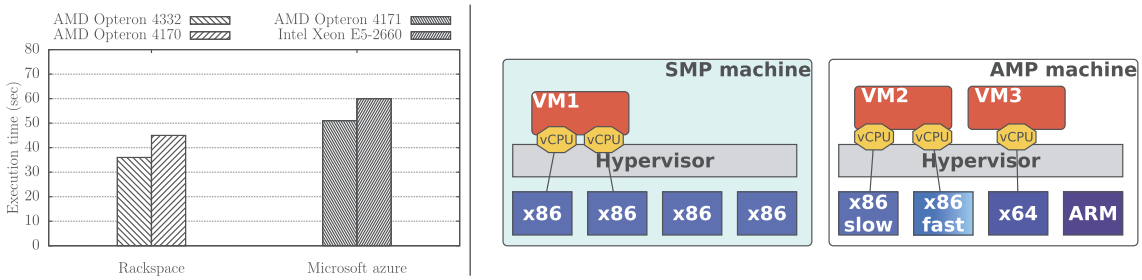

Fig. 1. Left figure: performance unpredictability illustration in Rackspace and Microsoft Azure clouds. The experimental application is π-app [6]. **Right figure:** SMP and AMP machines

The majority of research projects, if not all, have investigated this issue from the resource contention perspective [27,30], seen as the only source of the problem. However, we have shown in a previous work [28] that heterogeneity (of memory and processor) is actively involved in performance unpredictability. In this previous work, we focused on heterogeneous Symmetric Multicores Processor machines (hereafter called SMP clouds) (Fig. 1 left). However, advances in semiconductor technologies have enabled processor manufacturers to integrate more and more cores on a chip. This will lead in the near future (for energy saving reasons [8,18]) to a new type of architecture called Asymmetric Multicores Processor (AMP) (Fig. 1 right). Such an architecture is composed of cores exporting the same Instruction Set Architecture (ISA) but delivering different performance [18]. This new architecture comes with new challenges which have begun to be studied [8,14]. This paper tackles the issue of performance unpredictability in AMP clouds in which three problematic situations can be identified:

1. A multi-vCPU VM whose vCPUs run atop different core types (e.g. in Fig. 1 right, a thread inside the $VM2$ can be scheduled either on "vCPU $\times$86 slow" or "vCPU $\times$86 fast");

2. the scheduling of one vCPU across different core types in the same machine (e.g. in Fig. 1 right, $VM3$'s vCPU can be scheduled either on "×86 slow", "×86 fast", "×64" or "ARM");
3. VM migration across different machine types.

The first two situations can only occur in an AMP cloud while the third situation can raise up in AMP and SMP clouds.

It is clear that providing the same vCPU computing capacity regardless the underlying core type allows addressing all of the above situations. Our analysis of this problem led us to the design of two solutions, which both include (1) an absolute metric to express a vCPU computing capacity, and (2) a scheduler which enforces the negotiated contract during the overall VM lifetime. The first solution consists in using a reference core (noted p_{ref}) as the basis of vCPU capacity expression. Relying on the proportionality coefficient between the actual core type and p_{ref}, the scheduler dynamically adjusts the allowable CPU time of the vCPU. This solution is an improvement of our previous work [28] (which was performing such an adjustment at VM migration time in an SMP cloud). The second solution uses the "number of instructions per second" (noted Ips) as the metric to express a vCPU computing capacity. It requires a new kind of scheduler which relies on the actual number of CPU retired instructions rather than the CPU time (the standard practice). Theses solutions were prototyped in the Xen 4.2.0 system, although their design is independent from any virtualization system. The overhead of these prototypes at runtime is almost nil. We have evaluated their effectiveness using well known benchmarks (SPEC CPU2006 [4], Blast [1], and wordpress [5]). The evaluation results show that our solutions almost cancel out the issue of performance unpredictability due to core heterogeneity.

The rest of the article is structured as follows. Section 2 presents the background. Section 3 presents our contributions. The evaluation results are reported in Sect. 4. The related work is presented in Sect. 5 and we present our conclusions in Sect. 6.

2 Background

Our work is based on the para-virtualized Xen system. Before going into the description of our contributions, we briefly present Xen and its CPU allocation mechanism.

2.1 The Xen Hypervisor

Xen [11] is a popular open-source Virtual Machine Monitor (VMM) system (also called *hypervisor*) which is widely espoused by several cloud providers such as Amazon EC2. Its implementation follows the para-virtualization [29] model. In this model, the hypervisor runs directly on the hardware, so taking the traditional place of the operating system (OS). Thus, the hypervisor has all privileges

and rights to access the entire hardware. It provides the means to concurrently run several OS called virtual machines (VM). The host OS (seen as a special VM) is called *dom0* while the others are called *domU*. The former has more privileges than the latter since it is responsible for running Xen's management toolstack. The next section presents the Xen's CPU allocation mechanism.

2.2 CPU Allocation in Xen

Each VM is configured at start time with a number of vCPU and the hypervisor is responsible for scheduling vCPUs on cores. Roughly, each core runs a dedicated scheduler instance which manages a sub-group of vCPUs. The goal of each scheduler is to determine which vCPU will receive the core during the next quantum. Xen implements several scheduling policies including Simple Earliest Deadline First (SEDF) [11] and Credit [11]. SEDF is a scheduler which guarantees a minimum processing time to a VM. Concerning the Credit scheduler, it guarantees that a VM will strictly receive a portion (called *credit*) of the physical machine computing capacity. Credit is the default and the widely used scheduler. Therefore our work only considers this scheduler.

The Credit scheduler works as follows. Each VM (noted v) is configured at start time with a credit value (noted c) between 0 and 100 (full computing capacity). The scheduler defines *remainCredit*, a scheduling variable (associated with the VM) initialized to c. Each time a vCPU from v releases a core, (1) the scheduler translates into a credit value (let us say *burntCredit*) the time spent by v on the core. Subsequently, (2) the scheduler computes a new value of *remainCredit* by subtracting *burntCredit* from the previous *remainCredit*. When *remainCredit* reaches a lower threshold (configured in Xen), the VM enters a "*blocked*" state. In order to make blocked VMs schedulable in the future, the scheduler periodically increases their *remainCredit* according to their initial credit.

From the above presentation, we can see that the Credit scheduler is based on the notion of credit which depends on CPU time. The latter is a relative metric, as opposed to absolute metrics introduced in Sect. 1. Indeed, a vCPU capacity during a time period depends on the underlying core type. In other words, during the same time period, different core types result in different numbers of retired instructions for the same application. The next section presents our solutions which address this issue.

3 Performance Predictability Enforcement Systems

In public clouds, a vCPU is generally pinned to a dedicated core and is allowed to fully use this core. Our work is situated in this context[1]. In such a context, the provider presents to the user the vCPU capacity as a core capacity. This is ambiguous in AMP clouds since cores have different capacities. This

[1] Our solutions can also be easily applied to other contexts where several vCPUs share the same core.

paper addresses the issue of performance unpredictability which comes from this ambiguity. To do so, we adopt a two-step approach which is summarized by the following questions:

- Expressiveness: how to clearly express a vCPU computing capacity?
- Enforcement: how to enforce a booked vCPU computing capacity at runtime?

This section presents two ways to answer the above questions. Relying on the popular open source Xen hypervisor, we also present the implementation of each solution.

3.1 The First Solution

Expressiveness. In this solution, a vCPU computing capacity is presented to the user as the capacity of a specific core type (referred to as "reference core" and noted p_{ref}) available in the IaaS. p_{ref} is chosen once by the provider. It should be the core type with the lowest computing capacity, so that all other core types are able to provide this capacity.

Enforcement. Let us note *app* a single-thread CPU bound application (e.g. π-app [6]). $ExecutionTime(app, p)$ is the execution time of *app* when it exclusively runs on a core whose type is p. The enforcement system goal here is to ensure that given a vCPU v, $ExecutionTime(app, p)$=$ExecutionTime(app, p_{ref})$ regardless the actual core which runs v. We define the proportionality coefficient between p_{ref} and each core type p (noted $coef(p)$) as follows

$$coef(p) = \frac{ExecutionTime(app, p_{ref})}{ExecutionTime(app, p)} \quad (1)$$

The proportionality coefficient is computed once by the provider. Then, the enforcement system relies on an adaptation of the Xen Credit scheduler in order to dynamically scale each vCPU allowable CPU time according to the proportionality coefficient of its actual core. By doing so, the computing capacity associated with a vCPU is always that of p_{ref}. In the scheduler, the burnt CPU time (for a vCPU) is always translated as if it had been executed on p_{ref}. This translation is periodically performed after each scheduler intervention (typically every 30 ms in Xen). Unlike the native Credit scheduler (see the beginning of the section) which allows the vCPU to fully use its actual core (noted p), our modification enforces the use of only a fraction of p. The implementation of this solution is straightforward in the Xen Credit scheduler. It simply consists in modifying the *vCPUBurntCredit* function (see Sect. 2) as follows

```
1Unsigned int vCPUBurntCredit(...){
2         ...
3         //burntCredit has been calculated above (in the ↘
              original Xen Credit scheduler)
4         burntCredit = burntCredit * coef(typeOf(core_id));
5         return burntCredit;
6}
```

where $typeOf(core\_id)$ returns the current core type.

3.2 The Second Solution

Expressiveness. In this solution, a vCPU computing capacity is presented to the user as an instruction throughput (noted Ips): it is the maximum number of instructions the vCPU is allowed to performed per second. As well as the metric used to express a virtual network card capacity (Byte per second, Bps) is clear and absolute, Ips is also clear and absolute.

Enforcement. The enforcement system aims at ensuring that a vCPU's booked Ips is always satisfied regardless its actual core speed. Unlike the first solution which relies on the translation of a relative metric into an absolute metric, the second solution is directly based on an absolute metric. Therefore, the implementation of this solution cannot be implemented with a simple adaptation of the Xen Credit scheduler. It requires a monitoring system which is able to measure online the number of instructions performed by each vCPU. Ips_Sched, the new scheduler we have implemented, works as follows. Ips_Sched periodically collects the number of retired instructions (noted ri) related to each $vCPU$ during the sampling period (noted sp). In our prototype, ri is obtained using Perfctr-xen [24], a tool which allows accessing performance counters in a virtualized environment. Subsequently, Ips_Sched computes the actual instruction throughput (noted act_t) of each $vCPU$ using the following formula

$$act\_t = \frac{old\_t \times sp + ri}{2 \times sp} \tag{2}$$

where old_t is the throughput calculated during the previous sampling period. Note that old_t is zero if the vCPU was blocked during the previous sampling period. Ips_Sched keeps two queues namely $UNDER$ and $OVER$. If act_t is lower than the booked Ips, the vCPU is inserted into the $UNDER$ queue. Otherwise the vCPU is inserted into $OVER$. vCPUs which belong to the latter are not allowed to use the processor during the next sampling period (they are considered as blocked).

3.3 Comparison of the Two Solutions

This section presents both the advantages and the limitations of our two solutions. We have conducted a survey of cloud users (from two cloud provider partners) regarding the metrics used in the two solutions. The results of this survey show that the metric introduced in the first solution (the vCPU capacity is that of a reference core, p_{ref}) is more comprehensive than the metric used in the second solution (the vCPU capacity is an instruction throughput, Ips). The latter is suitable for HPC cloud users since they have the necessary expertise needed to deal with low level statistics such as Ips. Furthermore, Ips allows doing both fine grained and flexible CPU reservation. For instance, in the same way that physical AMP machines exist, the user can define AMP VMs[2] by expressing different

[2] Several research have highlighted the benefits of AMP VMs for energy saving improvements.

Ips per vCPU for the same VM. This is not possible using the first solution since all vCPUs should have the same capacity. Finally the implementation of the first solution requires more work (calibration of proportionality coefficients) from the provider than the second solution.

4 Evaluations

This section presents the evaluation results of our solutions. We evaluate the following aspects:

- Effectiveness: the capacity of the solutions to ensure a vCPU computing capacity.
- Overhead: the amount of resources consumed by both solutions.

Experimental Setup.

Hardware. The adopted experimental environment is similar to those used in prior work [15,26] in the domain of AMP. In those works, an AMP machine consists of two core types namely fast and slow cores. An AMP machine is simulated by an SMP machine whose cores work at different frequency levels: a fast core is emulated by running the core at the highest available frequency; a slow core is emulated by running the core at the lowest available frequency. Our testbed is composed of 2 DELL PowerEdge R420 machines. Each machine has 2 sockets, 6 cores per socket. The core's highest frequency is 2.2 GHz and the lowest frequency is 1.2 GHz. Each socket is organized into 3 fast cores and 3 slow cores. The operating system is Ubuntu 12.04 (Linux kernel version 3.8.0) virtualized with Xen 4.2.0. Our private IaaS is managed by OpenStack [3], a popular IaaS manager system.

Benchmarks. We evaluated our solutions using three reference benchmarks namely SPEC CPU2006 [4], Blast [1] and wordpress [5].

- SPEC CPU2006 [4] is a suite of single-threaded applications, stressing a system's processor, memory subsystem and compiler.
- Blast [1] is a multi-threaded application which simulates a typical workload from a health institute.
- Wordpress [5] is a web application commonly deployed in the cloud. Its performance metrics are the throughput (req/sec) and the response time.

4.1 The Effectiveness

Methodology. Performance predictability is guaranteed when the same workload execution results in the same performance regardless the core type. This contract is respected in an SMP machine because cores are identical (obviously) and we avoid other sources of problem (e.g. resource contention [30]) in order to only focus on the issue related to core heterogeneity. Therefore, we first execute applications on SMP machines managed with the native Xen system (representing the

"baseline"). Afterwards, we run the same applications on AMP machines managed with the native Xen (def) and with our solutions (sol1 and sol2). Finally, we compare the obtained results: our solutions are effective if they provide the same results as the baseline. In addition, to highlight the criticality of the addressed issue, we evaluate the use of the native Xen system to manage AMP machines. Notice that each experiment is repeated several times. In the evaluation of the first solution (sol1), p_{ref} is set to the slow core type. Concerning the evaluation of the second solution, the booked Ips of any vCPU is set to 50 $Mega\ Ips$.

Results. The first experiment uses CPU bound applications (SPEC CPU2006 and Blast) to evaluate the effectiveness of our solutions. As well as SPEC CPU2006 and Blast are respectively single-threaded and multi-threaded applications, they were ran respectively in single-vCPU and four-vCPU VMs. Each application is the subject of several executions, so that all vCPU to core type mappings are experimented. Performance predictability is achieved if the execution time of an application is almost the same in all executions. Figure 2 contains box plots presenting the normalized execution time of each benchmark (normalized to the baseline). The height of the boxes corresponds to the performance variation between various executions of benchmarks. We can observe that our solutions (sol1 and sol2) lead to a unique execution time which is equal to the baseline execution time. This is not the case for the native Xen system (def). The latter results in up to five different execution times, which correspond to the various vCPU to core type mappings: SPEC CPU2006 and Blast applications have respectively two and five possible vCPU to core mappings.

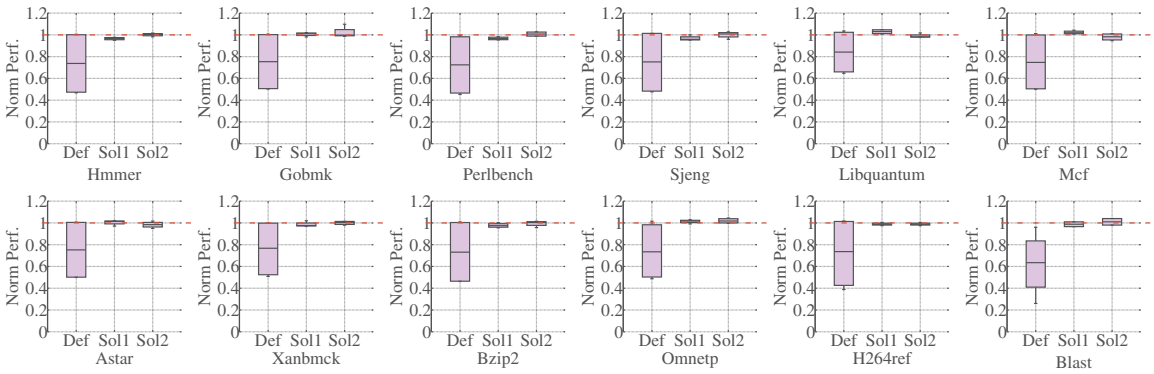

Fig. 2. Effectiveness evaluation of our solutions with SPEC CPU2006 and Blast.

The second experiment type is based on wordpress, an internet service application. We configure the benchmark as a two-tier application composed of a load balancer (Haproxy) which distributes requests among two Apache web servers (see Fig. 3 left). Each Apache server runs in a single-vCPU VM and a constant workload is submitted to wordpress. We experimented several vCPU to core type mappings (VM to core mappings). The results of this experiment are presented through a Cumulative Distribution Function (CDF) in Fig. 3 right. We

can observe that our solutions (sol1 and sol2) provide almost the same response time values as the baseline regardless the vCPU to core type mappings. Conversely, the native Xen (def) system results in two response time values (691 and 44385 micro sec) corresponding to the scenario where the two Apache servers (the two VMs) run on different core types.

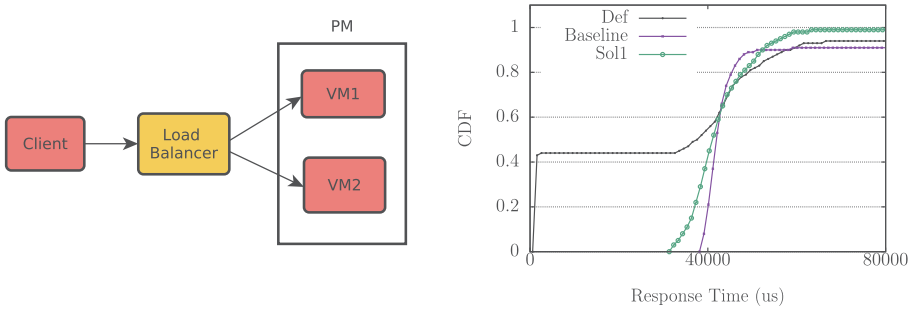

Fig. 3. Effectiveness evaluation of our solutions with wordpress. The right side figure presents the Cumulative Distribution Function (CDF) of the obtained response times.

4.2 The Overhead

The overhead of our solutions is almost nil. In reality, only the second solution could have introduced a possible overhead during performance counters collection. However, Perfctr-xen [24] authors and several other researches [10] have reported that this fear is unjustified. This also corresponds to what we have observed.

5 Related Work

The Heterogeneity Issue in SMP Clouds
Several researches have investigated the problem of hardware heterogeneity in today's clouds. [12] evaluates the impact of assuming a heterogeneous cloud as being homogeneous. It proposes a metric to express an application sensibility facing heterogeneity. [25] proposed to standardize the representation of the processing power of CPU by using Processing Units. [13] based on this Processing Units, presents the Execution and Resources Homogenization Architecture (ERHA). ERHA aims to provide mechanisms for submitting and executing batch applications in private IaaS clouds using homogeneous virtual environments created over heterogeneous physical infrastructure. Concerning public clouds, some (such as Amazon EC2) avoid the issue of hardware heterogeneity by dedicating the same hardware type to each VM type. For instance, EC2 announces to their customers that a m3.medium VM instance will always run atop an Intel Xeon

CPU E5-2650 2.00 GHz processor. This strategy is constraining for VM colocation. Indeed, a VM could not be deployed on a machine even if this machine has enough resources to host the VM. Concerning other public clouds such as Rackspace, the allocation unit is a vCPU and no more information is given about the real computing capacity. The actual computing capacity of a VM on this IaaS depends on the underlying core type, as illustrated in the introduction (see Fig. 1).

The Heterogeneity Issue in AMP Clouds
Several research works on AMP systems have focused on the scheduling issue and not on the predictability issue. Most of them have been conducted in the context of native systems. [9,17,18,21–23,26] aim at determining the best thread to core mapping in order to improve thread performance. [26] investigates applications which are composed of both parallel and sequential phases. [26] improves the scheduler by running sequential phase threads on fast cores. [7] tries to ensure fair sharing of the fast cores while [20] proposes to assigned vCPUs to core according to their speed. Therefore, fast core run-queues receive more vCPUs than slow cores. [9] shows that in an AMP, dynamic thread migration policies provide larger performance improvements than static policies. Their dynamic thread migration policy executes the threads for a small time duration on each core to measure their IPC (Instruction Per Cycle). Based on this, a thread that achieves only modest performance improvements from running on a fast core is executed on a slow core, and a thread that benefits significantly from running on a fast core is executed on the fast core. Researches conducted in virtualized systems [15,18,19,26] consist in translating native system solutions in virtualized systems (vCPUs are seen as threads). For instance, [15] proposes to realize a fair sharing of fast cores on AMP machines. They present a scheduling technique for hypervisors implemented in Xen. To ensure that all virtual CPUs (vCPUs) equally share the fast physical cores, the quota of a VM is decided depending on the number of vCPUs in it.

Positioning of Our Work
From far of our knowledge, no research study has investigated the issue of performance unpredictability in AMP clouds. The majority of research projects, if not all, have investigated this issue in SMP clouds. Also, they have mainly focused on the resource contention perspective, seen as the only source of the unpredictability problem. We have shown that heterogeneity (of memory and processor) is significantly involved in performance unpredictability. This paper is the only one to proposed solutions to the unpredictability in AMP clouds.

6 Conclusion

This paper addresses the issue of performance unpredictability due to the ambiguity of vCPU computing capacity expression in AMP clouds. We have presented two solutions and their implementations within the Xen virtualized system. Each solution includes both an absolute metric definition and an enforcement system.

The first solution relies on a reference core (p_{ref}) as the basis of vCPU capacity expression. Subsequently, relying on the proportionality coefficient between the actual core type and p_{ref}, the scheduler dynamically adjusts the allowable CPU time of the vCPU. The second solution directly introduces an absolute metric namely the "number of instructions per second" (noted Ips). We have demonstrated the effectiveness of each solution by experimenting several reference benchmarks.

References

1. Blast. http://fiehnlab.ucdavis.edu/staff/kind/Collector/Benchmark/Blast_Benchmark. Accessed 3 Feb 2015
2. Microsoft's top 10 business practices for environmentally sustainable data centers. http://www.microsoft.com/environment/news-and-resources/datacenter-best-practices.aspx. Accessed 10 Feb 2015
3. Open Stack. https://www.openstack.org/enterprise/virtualization-integration/. Accessed 3 Feb 2015
4. SPEC CPU2006. http://www.spec.org/cpu2006/. Accessed 3 Dec 2015
5. Wordpress. https://fr.wordpress.org/. Accessed 3 Feb 2015
6. y-cruncher - A multi-threaded Pi-program. http://www.numberworld.org/y-cruncher/#Benchmarks. Accessed 3 May 2014
7. Balakrishnan, S., Rajwar, R., Upton, M., Lai, K.: The impact of performance asymmetry in emerging multicore architectures. In: Proceedings of the 32Nd Annual International Symposium on Computer Architecture, ISCA 2005, pp. 506–517. IEEE Computer Society, Washington (2005). https://doi.org/10.1109/ISCA.2005.51
8. Baumann, A., Barham, P., Dagand, P.E., Harris, T., Isaacs, R., Peter, S., Roscoe, T., Schüpbach, A., Singhania, A.: The multikernel: a new OS architecture for scalable multicore systems. In: Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, SOSP 2009, pp. 29–44. ACM, New York (2009). http://doi.acm.org/10.1145/1629575.1629579
9. Becchi, M., Crowley, P.: Dynamic thread assignment on heterogeneous multiprocessor architectures. In: Proceedings of the 3rd Conference on Computing Frontiers, CF 2006, pp. 29–40. ACM, New York (2006). http://doi.acm.org/10.1145/1128022.1128029
10. Bui, V.Q.B., Teabe, B., Tchana, A., Hagimont, D.: Kyoto: applying the polluters pay principle to cache contention in an IaaS. In: Proceedings of the International Workshop on Virtualization Technologies, VT15, pp. 1–6. ACM, New York (2011). http://doi.acm.org/10.1145/2835075.2835077
11. Cherkasova, L., Gupta, D., Vahdat, A.: Comparison of the three CPU schedulers in Xen. SIGMETRICS Perform. Eval. Rev. **35**(2), 42–51. http://doi.acm.org/10.1145/1330555.1330556
12. Fedorova, A., Vengerov, D., Doucette, D.: Operating system scheduling on heterogeneous core systems. In: Proceedings of 2007 Operating System Support for Heterogeneous Multicore Architectures (2007)
13. Jin, X., Park, S., Sheng, T., Chen, R., Shan, Z., Zhou, Y.: ERHA: execution and resources homogenization architecture. In: The Third International Conference on Cloud Computing, GRIDs, and Virtualization, CLOUD COMPUTING (2015)

14. Jin, X., Park, S., Sheng, T., Chen, R., Shan, Z., Zhou, Y.: FTXen: making hypervisor resilient to hardware faults on relaxed cores. In: 21st IEEE International Symposium on High Performance Computer Architecture, HPCA 2015, Burlingame, CA, USA, 7–11 February 2015, pp. 451–462 (2015). https://doi.org/10.1109/HPCA.2015.7056054
15. Kazempour, V., Kamali, A., Fedorova, A.: AASH: an asymmetry-aware scheduler for hypervisors. SIGPLAN Not. **45**(7), 85–96. http://doi.acm.org/10.1145/1837854.1736011
16. Koh, Y., Knauerhase, R.C., Brett, P., Bowman, M., Wen, Z., Pu, C.: An analysis of performance interference effects in virtual environments. In: Proceedings of 2007 IEEE International Symposium on Performance Analysis of Systems and Software, San Jose, California, USA, 25–27 April 2007, pp. 200–209 (2007). https://doi.org/10.1109/ISPASS.2007.363750
17. Koufaty, D., Reddy, D., Hahn, S.: Bias scheduling in heterogeneous multi-core architectures. In: Proceedings of the 5th European Conference on Computer Systems, EuroSys 2010, pp. 125–138. ACM, New York (2010). http://doi.acm.org/10.1145/1755913.1755928
18. Kumar, R., Tullsen, D.M., Ranganathan, P., Jouppi, N.P., Farkas, K.I.: Single-ISA heterogeneous multi-core architectures for multithreaded workload performance. In: Proceedings of the 31st Annual International Symposium on Computer Architecture, ISCA 2004, pp. 64–77. IEEE Computer Society, Washington (2004). http://dl.acm.org/citation.cfm?id=998680.1006707
19. Kwon, Y., Kim, C., Maeng, S., Huh, J.: Virtualizing performance asymmetric multi-core systems. In: Proceedings of the 38th Annual International Symposium on Computer Architecture, ISCA 2011, pp. 45–56. ACM, New York (2011). http://doi.acm.org/10.1145/2000064.2000071
20. Li, T., Baumberger, D., Koufaty, D.A., Hahn, S.: Efficient operating system scheduling for performance-asymmetric multi-core architectures. In: Proceedings of the 2007 ACM/IEEE Conference on Supercomputing, SC 2007, pp. 1–11. ACM, New York (2007). http://doi.acm.org/10.1145/1362622.1362694
21. Liu, G., Park, J., Marculescu, D.: Dynamic thread mapping for high-performance, power-efficient heterogeneous many-core systems. In: 2013 IEEE 31st International Conference on Computer Design, ICCD 2013, Asheville, NC, USA, 6–9 October 2013, pp. 54–61 (2013). https://doi.org/10.1109/ICCD.2013.6657025
22. Luo, Y., Packirisamy, V., Hsu, W.C., Zhai, A.: Energy efficient speculative threads: Dynamic thread allocation in same-ISA heterogeneous multicore systems. In: Proceedings of the 19th International Conference on Parallel Architectures and Compilation Techniques, PACT 2010, pp. 453–464. ACM, New York (2010). http://doi.acm.org/10.1145/1854273.1854329
23. Morad, T.Y., Kolodny, A., Weiser, U.C.: Scheduling multiple multithreaded applications on asymmetric and symmetric chip multiprocessors. In: Third International Symposium on Parallel Architectures, Algorithms and Programming, PAAP 2010, Dalian, China, 18–20, pp. 65–72 (2010). https://doi.org/10.1109/PAAP.2010.50
24. Nikolaev, R., Back, G.: Perfctr-Xen: a framework for performance counter virtualization. In: Proceedings of the 7th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments, VEE 2011, pp. 15–26. ACM, New York (2011). http://doi.acm.org/10.1145/1952682.1952687

25. Rego, P.A.L., Coutinho, E.F., Gomes, D.G., de Souza, J.N.: FairCPU: architecture for allocation of virtual machines using processing features. In: Proceedings of the 2011 Fourth IEEE International Conference on Utility and Cloud Computing, UCC 2011, pp. 371–376. IEEE Computer Society, Washington (2011). http://dx.doi.org/10.1109/UCC.2011.62
26. Shelepov, D., Saez Alcaide, J.C., Jeffery, S., Fedorova, A., Perez, N., Huang, Z.F., Blagodurov, S., Kumar, V.: HASS: a scheduler for heterogeneous multicore systems. SIGOPS Oper. Syst. Rev. **43**(2), 66–75. http://doi.acm.org/10.1145/1531793.1531804
27. Tang, L., Mars, J., Soffa, M.L.: Contentiousness vs. sensitivity: improving contention aware runtime systems on multicore architectures. In: Proceedings of the 1st International Workshop on Adaptive Self-Tuning Computing Systems for the Exaflop Era, EXADAPT 2011, pp. 12–21. ACM, New York (2011). http://doi.acm.org/10.1145/2000417.2000419
28. Teabe, B., Tchana, A., Hagimont, D.: Enforcing CPU allocation in a heterogeneous IaaS. Future Gener. Comput. Syst. **53**(C), 1–12. http://dx.doi.org/10.1016/j.future.2015.05.013
29. Whitaker, A., Shaw, M., Gribble, S.D.: Scale and performance in the Denali isolation kernel. In: Proceedings of the 5th Symposium on Operating Systems Design and implementation Copyright Restrictions Prevent ACM from Being Able to Make the PDFs for This Conference Available for Downloading, OSDI 2002, pp. 195–209. USENIX Association, Berkeley (2002). http://dl.acm.org/citation.cfm?id=1060289.1060308
30. Zhuravlev, S., Blagodurov, S., Fedorova, A.: Addressing shared resource contention in multicore processors via scheduling. In: Proceedings of the Fifteenth Edition of ASPLOS on Architectural Support for Programming Languages and Operating Systems, ASPLOS XV, pp. 129–142. ACM, New York (2010). http://doi.acm.org/10.1145/1736020.1736036