Mitigating Performance Unpredictability in Heterogeneous Clouds

Boris Teabe, Alain Tchana, Daniel Hagimont

Mitigating performance unpredictability in heterogeneous clouds

Boris Teabe, Alain Tchana, Daniel Hagimont
University of Toulouse, Toulouse, France. E-mail: first.last@enseeiht.fr

***Abstract*—The speed of a device may vary since (i) IaaS hardware infrastructures are increasingly heterogeneous and (ii) devices often have a dynamically adjusted speed in order to adapt their energy consumption according to the load. This paper addresses SLA enforcement in a IaaS which includes devices whose speed vary. We show that resource management should rely on an absolute value SLA specification (i.e., a performance metric which is independent from the device speed) and a dynamic translation of this SLA into actual allocations according to the device speed. Surprisingly, while disk or network resource allocations already integrate such a scheme, CPU does not. We propose a CPU resource management system which implements absolute CPU allocation and dynamically translates it into actual CPU allocations according to CPU speed. We demonstrate and evaluate the benefits of this resource management system.**

***Index Terms*—Heterogeneous architecture; DVFS; Cloud**

I. Introduction

The majority of cloud platforms implement the Infrastructure as a Service (IaaS) model. In this model, the provider deals with customers through virtual machines (VM). The provider exposes a catalog of VM types among which the customer can shop. Each VM type is characterized by its size (the capacity of each resource type). This size corresponds to the Service Level Agreement (SLA) which should be met by the provider. On his side, the provider is interested in saving energy. To this end, two main techniques are commonly used by providers: VM consolidation [30] (through live migration) and device speed scaling [19]. VM consolidation allows gathering VMs on a reduced set of machines so that unused machines are switched-off. Device speed scaling (for underloaded devices) is also a means to save energy as reducing a device speed generally reduces its power consumption. From a more abstract point of view, VM consolidation among heterogeneous machines and speed scaling on one machine can be both considered as changing the nature and the performance of the underlying hardware.

Given a resource type (processor, main memory, etc.), a booked capacity may be expressed with relative values (relative to the hardware, i.e., a fraction of a device capacity, e.g., 30% of a device) or absolute values (i.e., a performance metric which is independent from any hardware, e.g., a throughput). Booking a resource capacity using relative values can be problematic since a device capacity may change as a consequence of the two above techniques. **The negotiated SLA should not vary according to energy management decisions.** For instance, a computing capacity which is expressed in terms of GHz is relative to the architecture of the target core (e.g. Out-of-Order architecture). Fig. 1 shows the execution time of π-app [22] (a CPU intensive benchmark) on two machine types (DELL and HP, the experimental context is presented in Section II-A) when their cores are configured to the same frequency level (1.6GHz). We can see a performance difference of about 21%. One would have expected the same performance because of the same frequency.

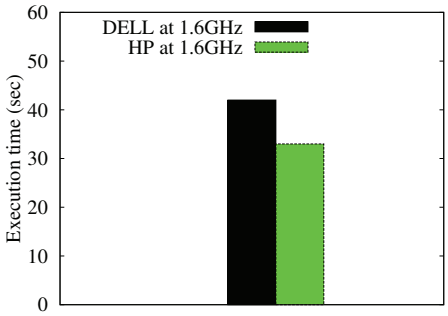


Fig. 1: The GHz is a relative metric.

Surprisingly, while VM disk or network capacities are booked using absolute values (bandwidth), the VM computing capacity (processor) is most of the time expressed with relative values [3], [4], [5]. In fact, today's VM schedulers such Xen credit and VMware are based on relative values. This leads to three problems:

- SLA breakage. It occurs when the actual computing capacity assigned to a VM by the scheduler is smaller than what the customer has booked.
- Performance unpredictability. It occurs when a VM

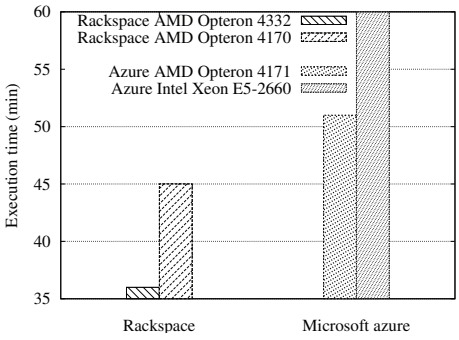


Fig. 2: Performance unpredictability on Rackspace and Microsoft Azure clouds for a Hadoop job.

is migrated (according to the VM consolidation strategy) across several machine types or when the Dynamic Voltage Frequency Scaling (DVFS) continually changes core frequency. Fig. 2 shows that the same VM type from Rackspace and Azure clouds delivers different performance level according to the underlying processor. This leads to the performance unpredictability problem [6], identified by Microsoft [8] as part of the five top significant challenges in the cloud. [7] has recently highlighted this issue in Amazon EC2 and Google Compute Engine too.

- Resource/money waste. It occurs when the actual computing capacity assigned to a VM by the scheduler is greater than what the customer has booked.

Section II illustrates these problems in detail. This article aims at addressing the issue of computing capacity allocation in a heterogeneous IaaS in which both VM migration and device speed scaling are used for energy saving. Since current schedulers fail to implement a truly absolute value based solution, we propose an absolute allocation unit. Then we show how such absolute allocations can be dynamically translated into relative values, which are well understood by today's schedulers [3]. Such translations take place when a VM is migrated or when a core frequency is changed. We implement a prototype in the Xen credit scheduler and we evaluate its benefits. Overall, the paper makes the following contributions:

- We demonstrate the issues related to heterogeneous IaaS (see Section II).
- We propose a solution to these issues (see Section III). This solution includes both a resource selling model and a generic implementation of this model. We present a prototype which is based on a popular virtualization system.

(P_0) DELL	Intel Core 2 Duo CPU E7300 @ 2.66GHz	Ubuntu 12.04
(P_1) HP	Intel Core i7-3770 CPU @ 3.40GHz	Ubuntu 12.04

TABLE I: Experimental PMs

- We perform intensive experiments using several reference benchmarks (see Section IV). The results of these experiments prove the effectiveness of our solution.

The end of this article presents both the related work (see Section V) and the future work (see Section VI).

II. Problem assessment

This section illustrates the issues we address in this paper.

A. Experimental context

The experimental cluster is composed of two machine types whose characteristics are presented in Table I. The virtualization system is Xen version 4.2.0, whose default scheduler is Credit [3] (see Section III-C). In order to facilitate the illustration, each PM is configured with a single core. The VM catalog exposes two VM types noted $VM20$ and $VM70$. A $VM20$ instance is allowed to use up to 20% of the computing capacity of its hosting machine. Concerning a $VM70$ instance, it is allowed to use up to 70%. Each VM instance is configured with a single virtual CPU (vCPU). It runs a CPU intensive application which consumes its entire capacity. The DVFS is provided by the on-demand governor [23] which adjusts a core frequency according to the load. We assume that reducing a core's speed slows down its computing capacity by 50%.

B. Assessment scenario

Let us consider a customer who requests four VMs noted $VM20_1$, $VM70_1$, $VM20_2$, and $VM70_1$. The IaaS manager starts $VM20_1$ and $VM70_1$ on P_0 while $VM20_2$ and $VM70_2$ are started on P_1. Fig. 3 presents the workload which is run by each VM. $VM70_1$ and $VM20_2$ end their job respectively at time "a" and "b". The VM consolidation strategy implemented in the IaaS works as follows. If a machine's CPU consumption is lower than 5%, all VMs on this machine are migrated to a machine which can host them. Therefore, both $VM20_1$ and $VM\ 70_1$ are migrated from P_0 to P_1 at time "c".

C. Assessment results

Fig. 4 presents the monitored load of each VM. It is interpreted as follows. First, the governor has decreased P_0 's speed at time "a" because P_0's global utilization falls under the DVFS threshold. This operation results in performance degradation on $VM20_1$: its second

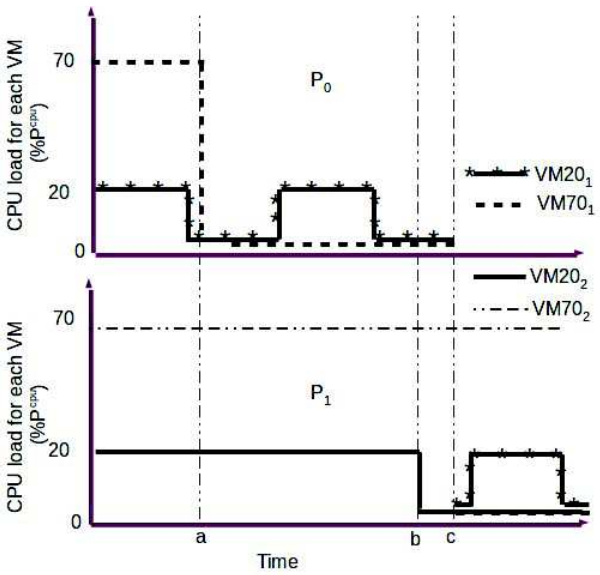


Fig. 3: Illustration scenario

peak load phase is larger than the first one (which is the expected duration). The SLA is broken. Second, $VM20_1$ and $VM20_2$ which are from the same type, have different performance because they run on different machine types. For instance, $VM20_2$ ends its jobs earlier (before time "b") than expected. The same goes for $VM70_1$ and $VM70_2$. P_1 is more powerful than P_0 (see Fig. 1). The same phenomenon is observed after time "c" when $VM20_1$ is migrated to P_1 ($VM20_1$ last peak is shorter than the first one). This situation leads to both performance unpredictability (the same VM type results in different performance) and resource waste (the provider could have satisfy $VM20_2$ and $VM70_2$ needs with less resources).

III. Contributions

This paper addresses the issues raised by CPU allocation when dealing with different core types in the IaaS. We first formalize both the functioning of a IaaS machine and the selling model we consider throughout this article. Subsequently, we present a resource allocation system which is based on absolute values. An implementation of this system in Xen is presented at the end of this section.

A. Formalization

1) The IaaS model: The IaaS is composed of m different machine types denoted by $\mathcal{P}_i$ ($1 \leq i \leq m$) and P_i is a type $\mathcal{P}_i$ machine. According to the DVFS activity, a machine's core can run at different frequencies. Therefore, the notation $P_{i(f)}$ means that P_i's core

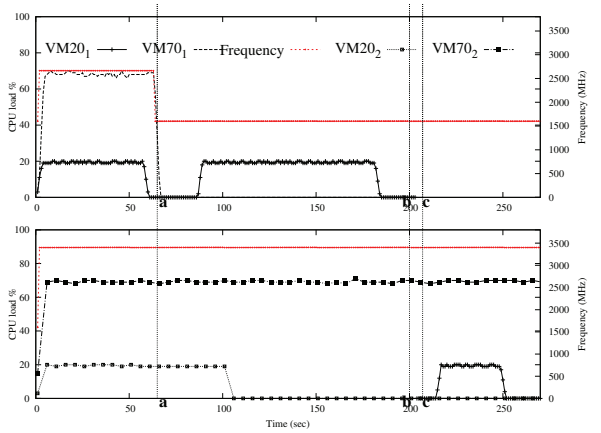


Fig. 4: The effects of CPU allocation based on relative values.

frequency is actually set to f. The maximum frequency value is noted max. From a more abstract point of view, we can say that $\mathcal{P}_{i(f)}$, $\mathcal{P}_{i(f')}$ and $\mathcal{P}_j$ are different machine types. Therefore, given a VM, both its migration (across different machine types) and frequency scaling are operations which modify its hosting machine type.

2) The resource selling model: VM computing capacities are discrete values which form a catalog of VM types (e.g. small, medium, large, etc.). This is the common practice in the cloud. Therefore, let us consider a customer who books a VM whose computing capacity is C_b. We claim that the provider should not use a machine dependent metric (e.g. GHz) when dealing with the customer. This is because the machine type which will run the VM is not known at negotiation time. Furthermore, it may change during the VM lifetime. We propose a negotiation metric which is independent from any machine type. It is called "Virtual Unit (VU)". In order to make things understandable by the customer, the computing capacity of a VU is indicated in terms of the execution time of a well known CPU intensive benchmark (e.g. SPLASH-2 LU [25]). Therefore, C_b is a number of VUs ($C_b = n \times VU$). Knowing that cores are shared among VMs, the main questions we need to answer is the following: How to translate a VU onto real CPU times and how to guarantee the constancy of this translation?

B. VU translation onto real CPU times

The translation system is based on a unique reference machine type, noted $\mathcal{P}_{ref}$. In fact, we act as the first machine which hosts any VM is a type $\mathcal{P}_{ref}$ machine. The latter is arbitrarily chosen by the provider and services as the foundation of all VU translation. Therefore, we

define a VU computing capacity as a fraction (noted c) of a P_{ref} computing capacity. From now on, such a fraction is called a credit. Therefore, any booked C_b corresponds to a P_{ref} credit noted C ($C = c \times n$). The main challenge now is to take into account the heterogeneous aspect of the IaaS. This is done by adapting C according to:

- the actual machine type which runs the VM;
- and its actual frequency.

Consequently, we propose a two-step credit adaptation system. The latter relies on the calibration of the IaaS.

1) Calibration: The calibration consists in determining the execution time of the CPU intensive benchmark mentioned above. This is performed for each VM type (thus credit) and for each machine type, taken at its maximum frequency. We note $T^{C'}_{i(max)}$ the execution time of the benchmark atop a $P_{i(max)}$ with the credit C'.

2) First-step credit adaptation: If C is the VM credit on $P_{ref(max)}$, and P_i is the machine chosen by the IaaS manager to host the VM (at creation or migration time), we first compute C', the credit to assign to the VM on $P_{i(max)}$. To do so, we rely on the calibration results. C' is chosen so that the following equation is respected:

$$T^{C'}_{i(max)} = T^{C}_{ref(max)} \tag{1}$$

3) Second-step credit adaptation: Each time a machine speed is modified (set to $P_{i(cur)}$), we need to recompute all VM credits. Let us note C" the new credit. C" is computed so that the following equation is respected:

$$T^{C"}_{i(cur)} = T^{C'}_{i(max)} = T^{C}_{ref(max)} \tag{2}$$

This equation summarizes the way we implement an absolute metric. It relies on two main assumptions (which have been demonstrated in our previous work [4], [5])

Frequency and performance proportionality: This property means that if we modify a core frequency, the impact on a VM performance is proportional to the change. It is defined by:

$$\frac{T^{C'}_{i(max)}}{T^{C'}_{i(cur)}} = \frac{f^{cur}_i}{f^{max}_i} \tag{3}$$

We define the frequency ratio on P_i as $ratio_{i(cur)} = \frac{f^{cur}_i}{f^{max}_i}$.

Credit and performance proportionality: This property means that if we modify a VM credit, the impact on its performance is proportional to the change. It is defined by:

$$\frac{T^{C'}_{i(f)}}{T^{C"}_{i(f)}} = \frac{C"}{C'} \tag{4}$$

Let us get back to the purpose of the second-step credit adaptation system: the computation of C", the new VM credit which takes into account the actual core frequency ($P_{i(cur)}$).

From the equation 4, $C" = \frac{T^{C'}_{i(cur)} \times C'}{T^{C"}_{i(cur)}}$.

From the equation 3, $T^{C'}_{i(cur)} = \frac{T^{C'}_{i(max)}}{ratio_{i(cur)}}$, and $T^{C"}_{i(cur)} = \frac{T^{C"}_{i(max)}}{ratio_{i(cur)}}$ which means that

$$C" = \frac{T^{C'}_{i(max)} \times C'}{T^{C"}_{i(max)}} \tag{5}$$

we want $T^{C"}_{i(cur)} = T^{C'}_{i(max)}$ (see the equation (2)). Therefore, C" is given by the following equation

$$C" = \frac{C'}{ratio_{i(cur)}} \tag{6}$$

C. Implementation in Xen

We have implemented a prototype of our solution in Xen hypervisor (version 4.2.0), more precisely its scheduler. Xen supports several schedulers [3] among which Credit is the default and the most used one. Therefore, our prototype is based on this scheduler. Credit is a proportional fair share conserving scheduler. Without describing it in detail (one could refer to [3]), let us give a general overview. In this scheduler, each VM v is assigned a credit c at creation time. This credit represents the maximum CPU time the VM is allowed to use. The scheduler defines *remainCredit*, a scheduling variable initialized with c. Each time a v's vCPU is scheduled on a core, the scheduler performs two main operations. (1) Firstly, it translates into a credit value the time spent by the vCPU on the core, this time is called *burntCredit*. (2) Subsequently, it computes a new value for *remainCredit* by subtracting *burntCredit*. If *remainCredit* reaches a lower threshold (periodically computed by the scheduler), the VM is no longer allowed to use a core. The scheduler periodically increases the value of *remainCredit* for each non schedulable VM in order to give them a possiblity to become schedulable.

We mainly describe the implementation of the second-step of our approach because, the first-step is realised easily by the IaaS provider (or the IaaS managing system) when he starts or migrates VMs. Regarding the second-step of our solution, the extension we made is straightforward. It consists in adjusting *burntCredit* according to the actual core frequency (see equation 6). This means inserting a new operation (let us say $adjustBurntCredit$) between step (1) and step (2).

The following algorithm presents in a pseudo code the implementation of $adjustBurntCredit$.

```
1Unsigned int adjustBurntCredit(unsigned int ↘
     burntCredit, int cpuID){
2        int freq=getCpuFreq(cpuID);
3        double ratio=freq/Freq[fmax];
4        burntCredit=burntCredit*ratio;
5        return burntCredit;
6}
```

It takes as input two parameters: the value of *burntCredit* as computed by the original Credit scheduler, and the actual core identifier. As one can intuitively imagine, this implementation incurs a negligible overhead. It also scales very well. Indeed, the complexity of our solution is O(#VMs). Knowing that the number of VMs a machine can simultaneously host is not usually high, the CPU time required by our solution is negligible.

IV. Evaluations

This section presents the evaluation of our solution. Micro-benchmarks are used to validate internal mechanisms while complex benchmarks show the effectiveness of our solution on realistic applications. The experimental context is the same as presented in Section II-A with P_0 be P_{ref}. SPLASH-2 LU [25] has provided the calibration benchmark.

A. Evaluation with micro-benchmarks

We have replayed the scenario presented in Section II in which the Credit scheduler is replaced by our scheduler. Fig. 5 presents the results of these experiments. The leftmost curves show the expected results (the baseline). They correspond to the execution on $\mathcal{P}_{ref}$ type machines in which the DVFS is disabled. The rightmost curves show the results of the experiment realized in a heterogeneous environment, as described in Section II. In comparison to what we have presented in Section II, the following observations can be made. First, the experiment runs within the expected time. $VM20_2$ and $VM70_2$ are assigned the appropriate credits on P_1 (resp. 11 and 40, because P_1 is more powerful than P_{ref}). This allows the provider to avoid resource waste. Secondly, $VM20_1$ is assigned more credits (from 20 to about 35) when the governor decreases P_{ref}'s speed (because $VM70_1$'s job ends at time "a"). Thirdly, when $VM20_1$ and $VM70_1$ are migrated to P_1 (at time "c"), their credits are recomputed. This explains the fact that the duration of $VM20_1$'s last peak is equal to the expected one. Finally, the sporadic peaks observed on the P_1 frequency curve are explained by the fact that P_1's CPU load is close to the DVFS threshold. In summary, we can see that our solution addresses the issues highlighted in Section II.

B. Evaluation with complex benchmarks

We evaluated our solution using complex benchmarks (not only CPU intensive applications). All machines are multi-core. Each experiment is played in two contexts: only on $\mathcal{P}_{ref}$ type machines in which the DVFS is disabled (the baseline) and on heterogeneous machines in which both the DVFS and a VM consolidation system is activated. We compare our solution with the native Xen system.

1) SPEC CPU2006: SPEC CPU2006 [28] is a widely used benchmark for measuring a hardware computing capacity. It consists of source code benchmarks which are developed from real user applications. These applications depend on the processor, the memory and the compiler of the tested system. Fig. 6 top presents the evaluation results of each SPEC CPU2006 application running in a VM70 type VM. We can see that the native Xen system still provides poor results (up to 42% of difference) while our solution tends to the baseline. In comparison with the results presented in the previous section, one may ask the origin of the gap (up to 15%) between our solution results and the baseline results. We claim that our solution is still correct. The gaps are explained by the fact that all SPEC CPU2006 applications are not only CPU intensive (as in the previous section). Some of them perform a lot of memory operations [24], and P_{ref} and P_1 have heterogeneous memory architectures. This explains the fact that gaps are from different height. From Fig. 6 bottom, we can see that the height of the gap is proportional to the application cache miss rate. Recall that our solution focuses on the issues related to core heterogeneity.

C. Optimizations for memory heterogeneity

As seen above, even if the processor is the most critical component, the heterogeneity of some memory components could also impact a VM performance. Concerning the Front Side Bus, its speed is already taken into account by the core speed. Therefore, it is not necessary to consider it here. Concerning the memory bus, it connects the northbridge and the RAM, serves for transporting data between the core and the main memory. Two machines which have different memory bus speed will lead a memory intensive application to different performance. Relying on the stream [26] benchmark, we have evaluated the memory bus speed of our testbed machines: 4823.1MB/sec for P_{ref} and 6566MB/sec for P_1. The rest of this section presents an amelioration of our solution in order to taking into account memory bus heterogeneity.

Our basic idea consists in adjusting C" (see Section III-B3) according to the memory bus speed dif-

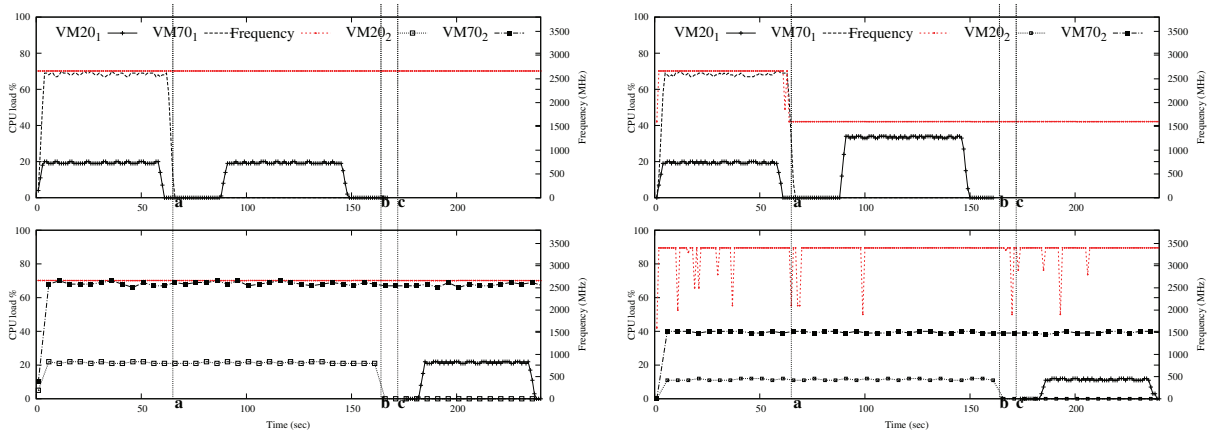
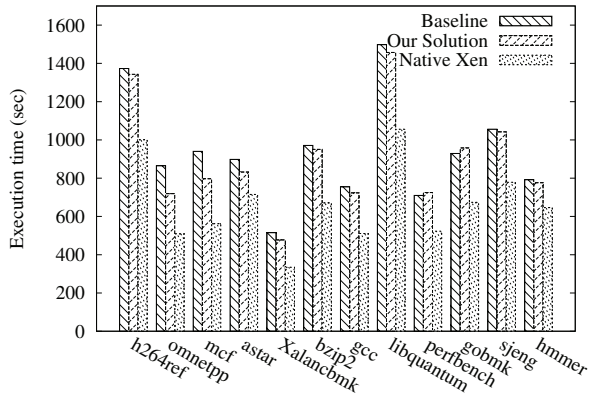


Fig. 5: Our solution (the two rightmost curves) provides the expected results (the two leftmost curves).



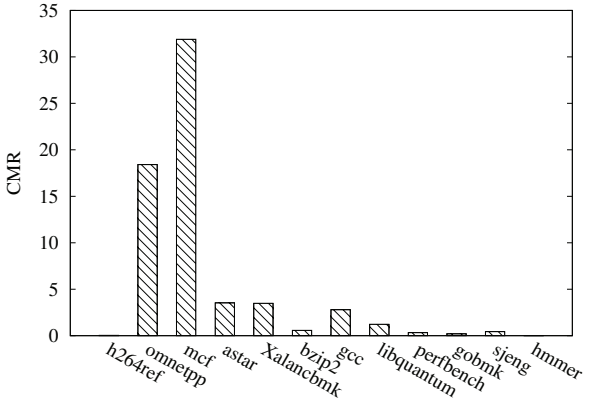


Fig. 6: (top) Evaluation with SPEC CPU2006. (bottom) Cache Miss Rate measurements

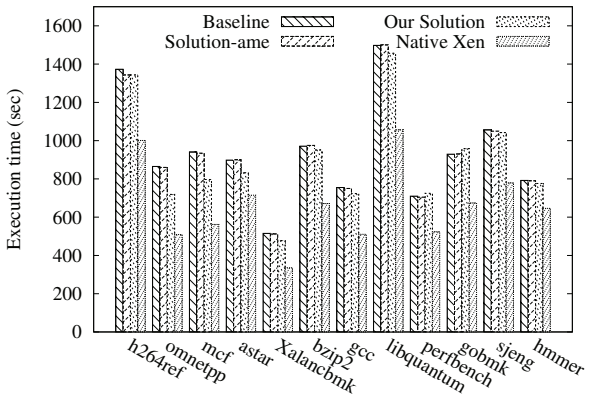


Fig. 7: Evaluation of the improved solution

ference between the reference machine and the actual machine which runs the VM. Therefore, the new credit (noted $C"_{ame}$) is computed as follows

$$C"_{ame} = C" * \frac{MB_{ref}}{MB_i} \quad (7)$$

where MB_i and MB_{ref} are respectively a P_i and a P_{ref} memory bus speed. This formula is only used when a VM runs a memory intensive application. In fact, as we have seen, CPU intensive VMs do not suffer from memory heterogeneity. Therefore, we have improved our implementation to periodically detect online each VM type (CPU or memory bound). This is archived by using performance monitoring unit (PMU) statistics such as cache miss rate and cache hits [13]. If the VM type is CPU bound, the equation 6 is used. The equation 7 is used otherwise. Fig. 7 presents the evaluation results of the improved solution (noted solution-ame).

V. Related work

Relative and absolute value based scheduling. Many scheduling algorithms use relative values [29] to allocate resources [1], [2], [3]. [9] studied IOPS reservation in a IaaS. It supports proportional-share fairness subject to a minimum and a maximum limit on IO allocations. It combines absolute and relative values in order to address workload fluctuation. Absolute values allow it to guarantee the minimum and the maximum limits. [10] did a similar work but it exclusively relies on relative values. Unlike [10], [9] may not suffer from the issues related to dynamic speed scaling and core heterogeneity.

Heterogeneity aware resource allocation. Several research have investigated the heterogeneity issue in shared hosting centers [11], [12]. Heterogeneity can be divided into two categories: hardware and workload heterogeneity. [14] evaluated the impact of assuming a homogeneous data center while it is heterogeneous. It proposes a metric to express an application sensibility facing heterogeneity.

Concerning public clouds, some avoid the issue of hardware heterogeneity by dedicating the same hardware type to each VM type. For instance, Amazon EC2 announces to their customers that a m3.medium VM instance will always run atop an Intel Xeon CPU E5-2650 2.00GHz processor. This strategy is constraining for VM consolidation. 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 its real computing capacity. The actual computing capacity of a VM on this cloud depends on the underlying core type.

[15] studied heterogeneity in EC2. Instead of providing a solution to guarantee performance, [15] proposed a gaming based placement which places VMs according to their EC2 analyses. [16], [17] investigated the same approach. [18] presented Paragon, a QoS-aware scheduling for heterogeneous workload in a datacenter. Its objective is to minimize performance degradation while we present a way to enforce an SLA.

Speed scaling aware resource scheduling. We have highlighted in a prior work [4], [5] the issues related to DVFS in the cloud. This work demonstrated the effectiveness of both frequency and credit proportionality mentioned in Section III-B3. Driven on the DVFS success for processors, [19] presented a DVFS solution for the memory. [20] presents an approach which combines service selection (replicated across many clusters of the same provider) and dynamic speed scaling in web service systems in order to achieve high energy efficiency while meeting performance requirements. [21] presented CoScale, a system which coordinates CPU and memory power management in order to improve energy savings compared to existing approaches which treat these devices separately. None of these work have tackled the issue of SLA enforcement in a cloud which includes both heterogeneous machines and variable speed devices.

VI. Conclusion

In this paper, we studied resource allocation in a IaaS environment. We showed that existing resource allocation systems which rely on relative values may lead to SLA violations in the context of a IaaS with heterogeneous machines or variable speed devices. While disk or network resource allocations are expressed with absolute values, CPU allocations are expressed with relative values (a percentage of a processor). We proposed an absolute allocation system for CPU and showed how it can be dynamically mapped onto physical resources. We implemented this solution in the Xen virtualization system and evaluated it in a private IaaS. These evaluations validated the effectiveness of our solution (no SLA violation).

Acknowledgements

This work benefited from the support of the French "Fonds national pour la Société Numérique" (FSN) through the OpenCloudware project

References

[1] Wei Jin, Jeffrey S. Chase, and Jasleen Kaur, "Interposed proportional sharing for a storage service utility", ACM SIGMETRICS Performance Evaluation Review 2004.

[2] Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica, "Dominant Resource Fairness: Fair Allocation of Multiple Resource Types", NSDI 2011.

[3] Ludmila Cherkasova, Diwaker Gupta, and Amin Vahdat, "Comparison of the Three CPU Schedulers in Xen," SIGMETRICS Performance Evaluation Review, 35(2) 2007.

[4] Teabe Boris, Tchana Alain, Daniel Hagimont,'Enforcing CPU allocation in a heterogeneous IaaS, Future Generation Computer Systems 2015

[5] Daniel Hagimont, Christine Mayap Kamga, Laurent Broto, Alain Tchana, Noel De Palma, 'DVFS Aware CPU Credit Enforcement in a Virtualized System', Middleware 2013.

[6] Younggyun Koh, Rob C. Knauerhase, Paul Brett, Mic Bowman, Zhihua Wen, and Calton Pu, 'An Analysis of Performance Interference Effects in Virtual Environments', ISPASS 2007.

[7] Christina Delmitrou and Christos Kozyrakis, 'HCloud: Resource-Efficient Provisioning in shared Cloud System', APLOS 2016.

[8] Microsofts Top 10 Business Practices for Environmentally Sustainable Data Centers, 'http://www.microsoft.com/environment/news-and-resources/datacenter-best-practices.aspx'.

[9] Ajay Gulati, Arif Merchant, and Peter Varman, "mClock: Handling Throughput Variability for Hypervisor IO Scheduling", OSDI 2010.

[10] Ajay Gulati, Irfan Ahmad, and Carl A. Waldspurger, "Parda: Proportional allocation of resources in distributed storage access", Usenix FAST 2009.

[11] Marco Canini, Vojin Jovanovic, Daniele Venzano, Dejan Novakovic, and Dejan Kosti, “Online testing of federated and heterogeneous distributed systems”, SIGCOMM 2012.
[12] Alexandra Fedorova, David Vengerov, and Daniel Doucette, “Operating System on Heterogeneous Core Systems”, ASPLOS 2013.
[13] Boris Teabe, Alain Tchana, and Daniel Hagimont, “Application-specific quantum for multi-core platform scheduler”, EuroSys 2016.
[14] Jason Mars and Lingjia Tang, “Whare-map: heterogeneity in ”homogeneous” warehouse-scale computers”, in ISCA 2013.
[15] Benjamin Farley, Venkatanathan Varadarajan, Kevin D. Bowers, Ari Juels, Thomas Ristenpart, and Michael M. Swift, “More for Your Money: Exploiting Performance Heterogeneity in Public Clouds”, SoCC 2012.
[16] Zhonghong Ou, Hao Zhuang, Jukka K. Nurminen, Antti Yla-Jaaski, and Pan Hu, “Exploiting Hardware Heterogeneity within the same Instance Type of Amazon EC2,” HotCloud 2012.
[17] Zhonghong Ou, Hao Zhuang, Andrey Lukyanenko, Jukka K. Nurminen, Pan Hu, Vladimir Mazalov, and Antti Yla-Jaaski, “Is the same Instance Type Created Equal? Exploiting Heterogeneity of Public Clouds,” TCC 2013.
[18] Christina Delimitrou and Christos Kozyrakis, “Paragon: QoS-Aware Scheduling for Heterogeneous Datacenters”, ASPLOS 2013.
[19] Qingyuan Deng, David Meisner, Luiz Ramos, Thomas F. Wenisch, and Ricardo Bianchini , “MemScale: Active low-power modes for main memory,” ASPLOS 2011.
[20] Jiwei Huang and Chuang Lin, “Agent-Based Green Web Service Selection and Dynamic Speed Scaling,” ICWS 2013.
[21] Qingyuan Deng, David Meisner, Abhishek Bhattacharjee, Thomas F. Wenisch, Ricardo Bianchini, “CoScale: Coordinating CPU and Memory System DVFS in Server Systems,” MICRO 2012.
[22] “y-cruncher A Multi-Threaded Pi-Program”, http://www.numberworld.org/y-cruncher/#Benchmarks, visited on September 2014.
[23] Venkatesh Pallipadi and Alexey Starikovskiy, “The ondemand governor: past, present and future,” Linux Symposium 2006.
[24] Sarah Bird, Aashish Phansalkar, Lizy K. John, Alex Mericas and Rajeev Indukuru, “Performance Characterization of SPEC CPU2006 Benchmarks on Intel Core 2 Duo Processor”, SPEC Benchmark Workshop, pp. 121-137, 2007.
[25] Steven Cameron Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta, “The splash-2 programs: Characterization and methodological considerations,” SIGARCH 1995.
[26] STREAM: Sustainable Memory Bandwidth in High Performance Computers. https://www.cs.virginia.edu/stream/.
[27] Tang, Lingjia, Jason Mars, and Mary Lou Soffa. “Contentiousness vs. Sensitivity: Improving Contention Aware Runtime Systems on Multicore Architectures,” EXADAPT 2011.
[28] SPEC CPU2006, http://www.spec.org/cpu2006/, visited on December 2015.
[29] Antonio Nicolo, *Efficiency and truthfulness with Leontief preferences. A note on two-agent, two-good economies*, Review of Economic Design 2004.
[30] Aziz Murtazaev and Sangyoon Oh. “Sercon: Server Consolidation Algorithm using Live Migration of Virtual Machines for Green Computing,” in IETE TR, vol. 28, issue 3, 2011.