Vertical scaling for prioritized VMs provisioning

Lenar Yazdanov
Computer Science Faculty
TU Dresden
Dresden, Germany
lenar@se.inf.tu-dresden.de

Christof Fetzer
Computer Science Faculty
TU Dresden
Dresden, Germany
christof.fetzer@tu-dresden.de

***Abstract*—The elastic scaling of services permits us (1) to meet service provisioning requirements (i.e., not to violate its SLAs), and (2) to efficiently use available resources (i.e., to minimize the cost of running the application). In this paper, we describe a novel online controller design built on top of the Xen hypervisor. The controller facilitates the elastic provisioning of applications using a combination of adapting the resource constraints of VMs and dynamically plugging in new virtual CPUs. We have implemented our design and evaluated our approach using real world workload traces from the RUBiS online auction benchmark. The results show that our controller allows to reduce total CPU time in comparison to statically allocated CPU with of minimum SLA violation rate and provides stable response time for high priority VM, when we apply it to provision prioritized VMs.**

***Index Terms*—Cloud Computing, Resource Scaling, Performance;**

I. Introduction

Clouds become a popular computing platform, which offers on demand computing resources and storage capacity. They allow to dynamically scale up and scale down underlying infrastructure in response to workload changes, performance requirements and other dynamic conditions. There are different mechanisms to implement auto-scaling. This mechanisms imply applying certain rules, which control scaling actions. Rules consist of conditions, which trigger this actions on infrastructure. Some systems [1], [2] provide users with conditions based on certain infrastructure metrics (e.g. CPU, memory). Others [3], [4] allow more complex metrics (combination of simple rules).

Such automation is complemented by well-known pay-as-you-go model. This pricing model eliminates initial costs of owning computational resources. User is charged hourly for the provisioned resources. However, challenges arise when we look deeper into mechanisms and cloud workload.

There are two types of scaling: horizontal and vertical. Horizontal scaling implies running new server replicas or load balancers to distribute load. While vertical scaling means changing on the fly assigned resources to a already running instance, for example letting more CPU to a running virtual machine (VM). Existing industrial platforms exploit horizontal scaling to automatically allocate resources during peak demand. The drawback of horizontal scaling is non-ignorable VM instance acquisition time. There are different numbers about the instance start-up lag[5], [6], but on average it takes 1 minute or more. In contrast, vertical scaling allows to double VM power within a second, which makes it attractive for applications with fluctuated behavior.

Cloud providers offer computing instances with fixed characteristics. Usually instances are not fully utilized by customer application. Some periods of time applications running inside VM require only half of allocated resources. Such behavior is common for web applications, which are driven by number of accesses per time unit. Therefore user overpays for allocated resources, when utilization is low. Additionally, same instance types do not expose the same performance characteristics [7]. Moreover, cloud user is charged by hour granularity pricing model. Hence, if VM runs only half an hour or even less, then user has to pay for the whole hour. Therefore existing provisioning and pricing schemes do not really serve the interests of customers.

Further investigation reveals, that cloud workload can be classified into two groups: time critical applications and applications with soft time requirements. First group consist of web applications such as internet stores, bookkeeping sites and etc. Responsiveness is essential for these applications, because slow respond leads to missing potential customers and revenue. Second group consists of applications running back-end tasks such as Hadoop MapReduce jobs and etc. This group of applications do not require real time responsiveness. This fact opens a way to resolve resource conflict using vertical scaling instead of triggering expensive horizontal scaling operations. Hence, when resource conflict happens an application of the first group can rent resources from an application of the second group. Therefore VMs need to be prioritized and have to have different prices. Moreover we think that shrinking rental durations from hour to second granularity compliments the idea of VMs prioritization and makes cloud more economically appealing for cloud user.

In this work, we investigate vertical scaling for provision web applications and resolving CPU resource conflicts. Our elasticity controller uses a prediction model to perform proactive resource scaling. We combine changing CPU power of virtual core with extending power of VM by plugging new CPUs on-the-fly. In contrast to horizontal scaling, vertical scaling allows to allocate resources with lower latency. Therefore, sudden workload increase does not cause severe SLA violation, which lets to delay slow operations of horizontal scaling. Moreover, we evaluate our implementation by running collocated VMs with different priorities to show vertical

scaling potential in resource conflict resolving.

Our paper makes following contributions:

- we introduce online elasticity controller, which allows to avoid SLA violation and provides mechanisms to efficiently allocate resources
- our approach exploits vertical scaling using combination of on-the-fly plugging CPU and tuning virtual CPU power
- we demonstrate efficiency of our controller applying real world workload traces of WorldCup 98
- we evaluate our controller to implement host resource conflict resolving
- we discuss benefits of applying vertical scaling to provision prioritized VMs.

The rest of the paper is structured as follows. In section 2 we describe vertical scaling design rationales. Section 3 describes implementation of elasticity controller. In section 4 we present our experiment setup and analyze results. Section 5 is dedicated to discussion about cloud pricing model. Finally we conclude the paper and present our ideas for future work.

II. Design rationale

Modern operating systems support CPU hot plug, i.e. a CPU could be plugged or unplugged to and from an already running VM. Recently, one of cloud providers [8] announced support of on-the-fly CPU plugging. This feature improves performance of CPU intensive applications. According to the presented benchmarks, CPU burstable VM (CPU is plugged-in to a running VM) essentially outperforms VM provided by EC2. Moreover, authors in [9] conclude, that a web server running inside vertically scaled VM outperforms load balanced web server running multiple VMs. Experiments show that vertically scaled VM reduces the provisioning overhead, mitigates SLA violations and maintains a higher throughput.

In addition to the feature provided by operating systems, existing virtualization technologies provide mechanism to implement vertical scaling on an already running VM. For example, *Xen credit scheduler* fixes the maximum amount of CPU a domain will be able to consume by setting a cap. The cap is expressed in percentage of one physical CPU: 100 is 1 physical CPU, 50 is half a CPU, 400 is 4 CPUs, etc [10].

Therefore CPU resources of VM can be tuned not only with number of running CPUs, but also by adjusting speed of virtual CPU as a fraction of physical CPU. In the next part of this section we describe our vision of scaling opportunities and give more details.

We think that efficient allocation of resources might be achieved by combining vertical and horizontal scaling together. Figure 1 describes scaling horizons of virtualized environment. There are different methods, which can be applied depending on scaling latency and scaling horizon. We define following scaling levels: VM, host and data center. VM and host levels belong to vertical scaling, and data center level belongs to horizontal scaling.

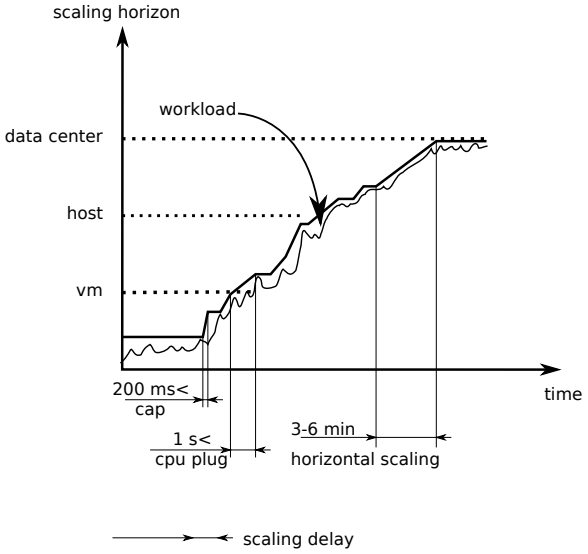

Fig. 1. Scaling latencies for different scaling horizons

A. VM level scaling

Applications running inside VM might not fully utilize allocated CPU resources all the time. One of examples is web applications, which utilization depends on a frequency of accesses. Therefore to follow utilization curve we apply capping. Authors in [11] state that changing the limit of a virtual machine CPU takes 120 ± 0.55 ms. Hence, capping re-allocates CPU resources with small overhead and low latency. However, the VM cannot get more than its maximum allocated resources even if a host has spare cycles on other cores.

B. Host level scaling

If VM level scaling reaches its maximum, then CPU-Hotplugging allows to extend a horizon for further virtual CPU tuning. Plugging CPU means that extended VM starts to use resource from collocated VM. Therefore it is recommended to prioritize VMs inside a host. For example, VMs can be prioritized according responsiveness requirements. Hence, depending on the priority, certain VM may get additional power, while other VMs pinned to a lower number of physical CPUs or their CPU share decreased. This scheme resolves CPU resource conflict. CPU-Hotplug on average takes less than 1 second. Hence, power of VM can be doubled immediately instead of waiting for start of new VM. Moreover, in comparison to horizontal scaling lightweight prediction mechanisms can be applied to plug a CPU.

C. Data center level scaling

We understand that vertical scaling bounded by maximum power of a host. Therefore horizontal scaling triggers, when resource conflict is impossible to solve inside the host. Necessity of this type of scaling may be proved with help of long term prediction or according daily patterns. Data center level scaling is part of our future research.

In order to implement efficient vertical scaling we combine capping and CPU-Hotplug. Consider the following example.

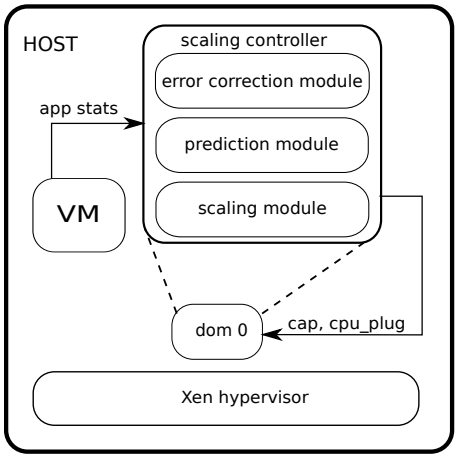

Fig. 2. Elasticity controller

Initially VMs get some fixed number of CPUs with certain power unit. VMs have different priority. The goal is to provide better performance for high priority VM. If load increases, then virtual CPU speed of high priority VM increases too. Once CPU power reaches its limit, additional CPU is plugged to this VM. Other low priority VMs use spare cycles of physical CPUs.

Having clients with different priorities is beneficial for a provider, because it allows to better utilize existing capacity and get additional income. Clients running high priority VMs get high availability, by renting resources from clients running low priority VMs. If high priority clients have spare resources, then they rent out these resources to low priority VM. We assume that cloud users aware of VM priorities and pay different prices. Therefore scheduling, described above, should not run in to severe SLA violation for one of low priority VMs.

III. Controller architecture

In this section we present our elasticity controller. We first describe architecture of our solution. Then we overview prediction model used by the controller. Next, we provide a description of error correction techniques for online resource provisioning.

A. Overview

Our scaling controller monitors resource usage of running VM. The architecture of controller is presented on figure 2. We use *xentop* resource usage monitor to get VM CPU utilization statistics. Monitoring is implemented with 1 second delay and thereby controller quickly detects changes in resource requirements and implements scaling immediately. CPU scaling is performed by adjusting CPU cap using *Xen credit-scheduler*. CPU plug is implemented using Xen's *vcpu-set* API. Used hypervisor version doesn't support CPU unplug, therefore we wrote small daemon which implements CPU ejection inside VM.

Every sampling interval CPU usage time series are sent to prediction module. This module performs forecasting of CPU usage value for the next step.

Error correction module adds an extra value to prediction value to avoid under-provisioning. Also controller has resource underestimation correction mechanism, which triggers according predefined threshold.

Scaling module sets CPU cap using value from previous modules. If predicted resource usage value exceeds current CPU power, then CPU plug action is triggered. Virtual core is unplugged when predicted value indicates that it is excessive.

B. Prediction module

1) Prediction: We use resource usage predictor evaluated in the [12]. Authors found that AR model order of 16 is the best linear prediction model to implement predictions of CPU usage from 1 to 30 seconds in the future. Moreover, it has low CPU cost - less than 10 ms for 2000 samples. Therefore it adds small overhead to CPU consumption of dom0. Our elasticity controller implements short-term one resource prediction. Therefore 100 recent CPU consumption samples are used as a training data.

C. Error correction module

Our controller performs active and proactive error correction. In the first case it corrects prediction error and in the second case it corrects under-estimation errors.

1) Prediction error correction: The goal is to avoid under provisioning. Therefore additionally prediction module implements error correction. Every sampling period the controller calculates prediction error $e_k < 0$, where e_k is underestimation error. Then the controller calculates *WMA* (weighted moving average) over last *k* values of underestimation prediction errors. This result is used as an extra value to predicted resource usage.

2) Resource usage based error correction: In conjunction with prediction based CPU allocation, the controller adapts to workload change using under estimation error correction. The main problem is that real resource demand is unknown during under-estimation. We have only lower bound. Therefore our controller incorporates relative resource utilization threshold, which triggers underestimation error correction. Relative resource utilization is defined with following equation :

$$r(k) = \frac{v(k)}{u(k)} \tag{1}$$

Where $v(k)$ is VM CPU usage and $u(k)$ is CPU allocation value, which is dynamically changed by controller.

Choosing appropriate threshold allows to provide high performance and avoid wasting of resources. This value is very much dependent on the type of workload. For example, highly interactive and bursty workloads require greater over provisioning value in comparison to less fluctuate workloads. Currently our scaling controller supports 90% threshold. Automatic determination of threshold is beyond the scope of this paper and will be part of future research.

Every sampling interval the controller compares relative utilization r with threshold r_{thr} to check if underestimation conditions are met. Resource error correction mechanism

triggers when CPU utilization exceeds this threshold. Error correction mechanism multiplies current cap value by coefficient α. This coefficient calculated in a following way :

$$\alpha = 1 + \frac{r - r_{thr}}{1 - r_{thr}} \quad (2)$$

D. Scaling module

Scaling module takes predicted value to perform resource allocation. The controller uses this value to fix the maximum amount of CPU a domain will be able to consume. Additionally module implements CPU plugging when predicted value exceeds current CPU allocation power:

$$CPU_{pred} > VCPUS_{currr} * 100 \quad (3)$$

where CPU_{pred} is CPU cap for the next step and $VCPUS_{currr}$ is the current number of virtual CPUs allocated to VM.

Our controller also implements CPU unplugging to release unused core. CPU scaling-down triggers by following equation:

$$CPU_{pred} > (VCPUS_{currr} - 1) * 100 \quad (4)$$

IV. Evaluation

A. Test bed

In [13] authors implemented benchmark which simulates the behavior of online bookstore such as *amazon.com*, bulletin board websites, and auction website. They conclude that the CPU of database tier of online bookstore is the bottleneck and CPU of web server is bottleneck for auction sites and bulletin board. Therefore we run our experiments on two tier web application using RUBiS online auction benchmark[14] to evaluate our controller.

The test bed consists of one computer which is used to run VMs and group of client emulator machines.

The computer running VMs has quad-core Xeon 2.66GHz, 16 GB memory, 100 Mbps network, and runs Ubuntu 11.10 with Xen 4.1. This machine has 3 guest VMs running CentOS 6.0 64 bit. One VM has 3 cores and runs DB server. Another VM is web server (Apache HTTP 2.2) which is managed by our online scaling controller. Third VM runs single node Hadoop, it has 2 cores and also is managed by the controller. Dom0 on the web tier machine is pinned to one core and does not share it with other VMs.

Workload generator consist of 10 machines with 100Mbps network, which run RUBiS benchmark client emulators.

We used web traces of WorldCup 98[15] to evaluate scaling controller with real workload variation. We apply this traces to RUBiS benchmark client emulator. Workload trace is constructed from six hours per minute request rate starting at 1998-05-10:03.00. Short workload trace is used to evaluate two VMs scaling.

Our evaluation primarily consists of two parts. First we implement single VM scaling, where we run web tier application. Controller scales web server VM. The goal is a comparison of static and dynamic provisioning policies and estimation of performance gained using our controller . We compare static CPU allocation with scaling schemes to express the benefit of online scaling controller. Experiments are implemented with following configurations: 1) *StUnder* : Static CPU allocation with under provisioning; 2) *StOver* : Static CPU allocation with over provisioning; 3) *OnlinePad* : Online CPU allocation with fixed padding, controller adds constant percentage value to predicted value; 4) *OnlineCorr* : Controller implements CPU allocation with resource usage error correction. For static resource allocation with under and over provisioning we run VM 1 and 2 virtual CPUs respectively.

In the second part we perform two VMs scaling and use short (16 minutes) real workload trace. One VM is a high priority VM (Web server) and another VM is a low priority VM, which runs Hadoop. Hadoop VM executes *Wordcount* application. These VMs share physical CPUs. The low priority VM gets spare physical CPU cycles which left from the high priority VM. Generally we compare four configurations: 1) *Standalone VM* : Web server VM runs alone, we do not run Hadoop jobs; 2) *Xen scheduler* : CPU allocation is implemented by standard *Xen credit scheduler*; 3) *Xen scheduler weighted* : CPU allocation is implemented by standard *Xen credit scheduler* running in weight mode, where the high priority VM gets 4 times higher weight than the low priority VM; 4) *Controller* : Our controller implements CPU allocation using *Xen credit scheduler* capping mode.

B. Results

1) Single VM scaling: On figure 3 is presented real CPU demand of RUBiS web server. CPU utilization fluctuates over time. Some hours have higher CPU demand, than others. Graph shows, that workload requires more than 1 CPU, because CPU usage some periods of time higher than 100%. Moreover, graph reflects real situation when cloud user provisioned VM with two CPUs and overpays for resources which are not used.

Figure 4 shows cumulative distribution of CPU utilization error prediction. The results show that CPU utilization prediction makes less than 3% significant($e > 5\%$) underestimation error.

Figures 5, 7, 6 show the efficiency of evaluated resource provisioning patterns. In the experiments SLA violation rate is set to 20 ms. CPU allocation time is calculated based on CPU cap value provided for evaluated schemes.

Figure 5 shows SLA violation rate for different policies. VM with 1 virtual CPU has highest SLA violation rate. It presents real situation of under provisioning. VM with 2 cores has a lowest SLA violation rate. Other results present SLA violation rates for scaling schemes. CPU allocation with padding achieves better result when higher padding value is used. The best results shows the error correction scaling scheme.

In figure 6 is presented response time of given schemes. All dynamic resource allocation models provide low mean response time. The best results between dynamic scaling schemes shows the error correction scheme, because it immediately reacts to under provisioning by allocating more

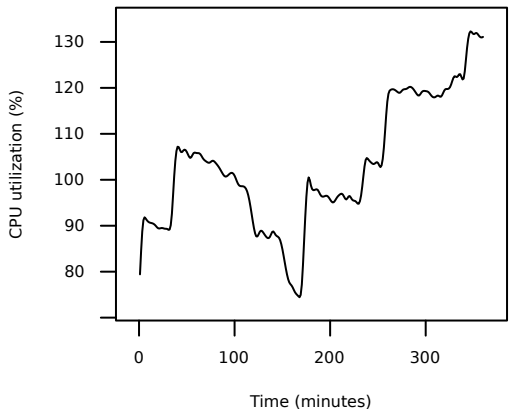

Fig. 3. Real CPU demand of the RUBiS webserver

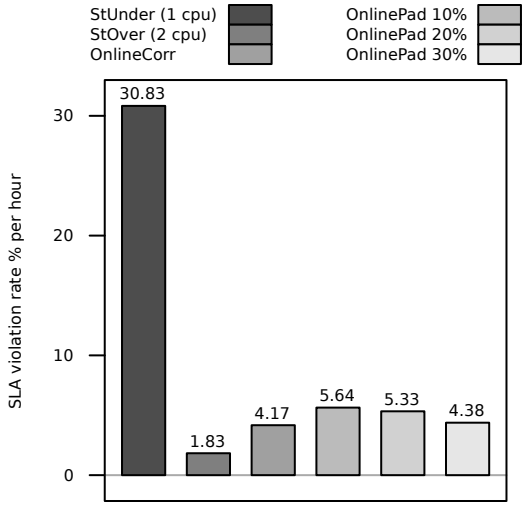

Fig. 5. SLA violation (response time $<$ 20 ms)

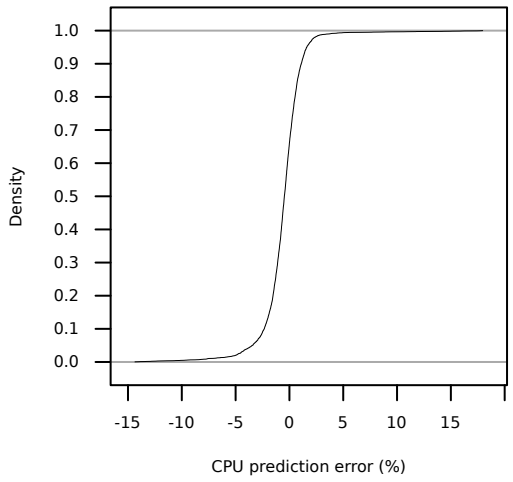

Fig. 4. CPU prediction error

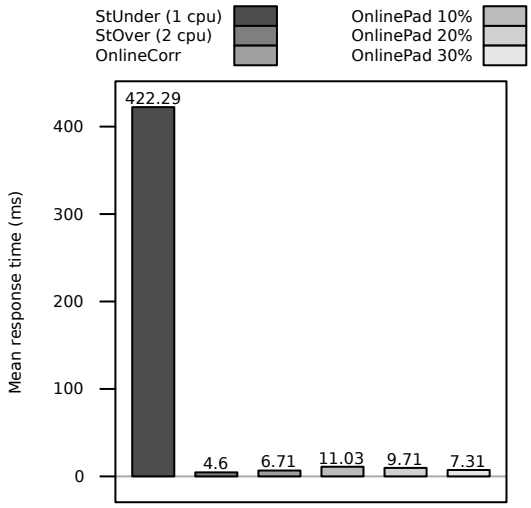

Fig. 6. Web server response time

resources.

Figure 7 shows total CPU allocation time. Static configuration with 2 virtual cores showed good performance, but requires 2 times more CPU in comparison to error consumption scheme. While the latter showed similar performance results and requires two times less CPU allocation. Therefore unused CPU cycles can be rent out to other VM to improve host resources usage efficiency.

2) Two VM scaling: In the experiment we used resource error correction scheme as the best between schemes evaluated in first part of experiments.

Figure 8 shows response time of web server VM sharing physical CPU with batch processing VM. Standard *Xen credit scheduler* does not guarantee stable responsiveness of high priority VM. Therefore response time fluctuates when batch processing takes a place. After 510 second Hadoop finishes its task and response time of web application becomes stable. In contrast to Xen scheduler our elasticity controller implements better scaling of high priority VM, which keeps response time stable. We also used weighted mode of *Xen credit scheduler* which allows statically assign VM priority. The graph shows that weighted mode provides better response time in comparison to non-weighted mode. However, the response time fluctuation is unstable during batch job execution and it is higher than response time achieved when we use our elasticity controller. Hence, capping mode allows to achieve better isolation in comparison to weighted mode.

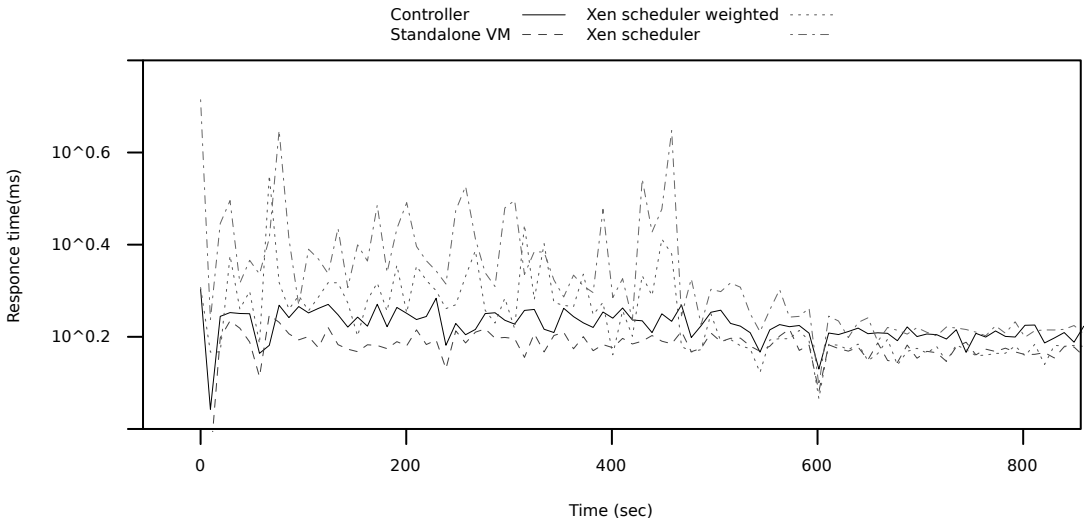

Fig. 8. Web server response time

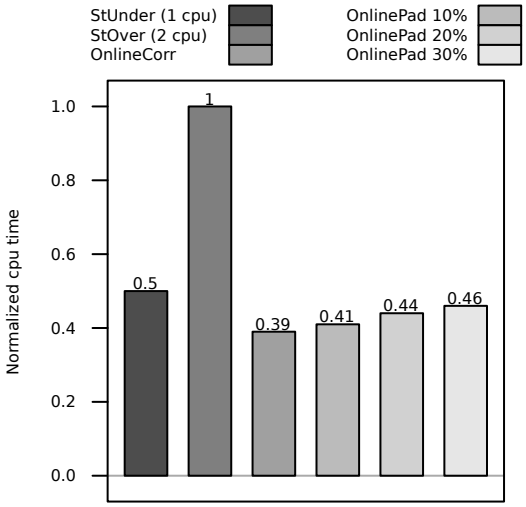

Fig. 7. CPU time

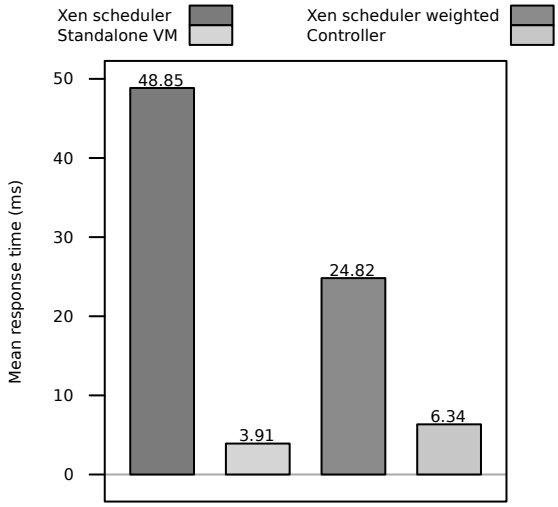

Fig. 9. Web server mean response time

Figure 9 shows web server mean response time of all evaluated configurations for first 510 seconds when batch processing VM runs Hadoop job. Our controller significantly outperforms *Xen scheduler*.

On figure 10 presented the execution time of batch job. CPU allocation implemented by *Xen credit scheduler* in non-weighted and weighted mode provides close execution time value. If we apply our controller then *Wordcount* application runs 2 times longer, than when we use *Xen scheduler*. However, if run time is not critical for low priority VM, then it is a small price paid to achieve stable response time of high priority VM.

V. Discussion

Amazon EC2 Spot Instances provide a way to get VM for a lower price. The idea behind the cost reduction is that provider has free capacity some periods of time. Spot Instances have lower price in comparison to on demand instance types, because provider does not guarantee execution time of spot instance. Hence, Amazon already has some kind of prioritization of VMs.

We think that further resource usage optimization may be achieved with help of vertically scaling VMs. Therefore VMs need to be explicitly prioritized. Customers depending on their applications workload type select VM with appropriate

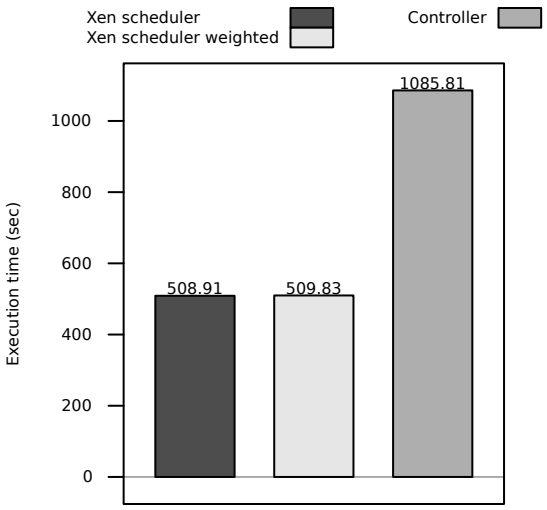

Fig. 10. Hadoop execution time

priority.

Another point is that prioritization requires shifting from hour granularity to second granularity. Hence, cloud provider rents out spare resources to low priority VM and later charges customer for consumed CPU time. This time is calculated according the cap value set to VM during certain second. The final price of used resources calculated as follows :

$$P_{total} = C * \sum_{i=}^{T} r_i \tag{5}$$

Where C is price of an hour of CPU time and r_i is CPU allocation during i-th second of VM execution time. If VM runs for T seconds, then customer according this equation pays for the actually used CPU resources.

Applying this cost model can emerge following service: Suppose a high priority VM which costs $0.3\$$ per hour and a low priority VM with price $0.05\$$ per hour. VMs run for 2 hours and the high priority VM during this time used 66% of CPU resources. Hence, the low priority VM used the rest 44%. The total cost for the high priority VM is $2hours * 0.66 * 0.3\$ = 3.96\$$ and for the the low priority VM is $2hours * 0.44 * 0.05\$ = 0.044\$$. Hence, in total cloud provider gets $3.96 + 0.044 = 4.004\$$

Now suppose current pricing model. Standard VM with CPU costs $0.1\$$ per hour. Customer, who wants to run web application and avoid under provisioning, selects VM with 2 CPU. Hence, total price of running this VM for 2 hours is $0.1\$ * 2CPUs * 2hours = 0.4\$$. Thus user overpays $0.4 - 3.96 = 0.04\$$. Moreover, there are yet an threat that future workload would require more than 2 CPUs, which would case under provisioning. If we look to the amount of money which pays user of low priority VM, then we notice that it is $0.1\$ * 2CPUs * 1hour = 0.2\$$. It requires to run VM only for 1 hour, because in prioritized VM case it used only 44%. It means that now low priority VM task uses 88% of 2 core instance during 1 hour. From total cost view customer overpays $0.2\$ - 0.04\$ = 0.16\$$, from used resources view this difference becomes higher.

Finally, we can estimate provider's revenue. Renting resource in case of prioritized VMs gives $4.004\$$ and current price model gives - $0.4\$$ for using the same capacity during two hours. Thus IaaS provider earns 0.1% more every 2 hours, if proposed pricing model applied.

We present here sample numbers, but in general this discussion example shows the advantage of applying vertical scaling to prioritized VMs provisioning. Applying proposed model is beneficial for all participants of cloud market. Provider gets an opportunity to better utilize it's capacity and get additional profit. Customer really pays only for used resources.

VI. Related work

There are numerous ongoing research in the field of resource allocation. Researchers apply different approaches to implement resource provisioning. This approaches involve the use of feedback controllers[16], regression analysis[17], analytical models[18]. This works present models which predict future workload, CPU utilization and response time for specific workload.

Amazon implemented Elastic Beanstalk - semi-automatic scaling system which allows to scale web application in case of high CPU consumption. User independently has to define trigger conditions which is difficult to figure out the proper scaling conditions if running inside VM application has fluctuated behavior. Moreover, system requires on average from to 3 to 6 minutes to get new VM up and running. This is essential latency for time-critical applications.

In [11] authors developed scaling system which performs online provisioning using predictions model. System changes virtual CPU allocation limit. If the system finds that in the future total load of all running VMs will exceed resource of the host, it triggers horizontal scaling. Authors omit the possibility of plugging CPU to high priority VM, which may improve overall performance without applying expensive horizontal scaling.

Previous work on vertical scaling [9] propose to use CPU hot plug to provision VM. Authors evaluated web application tier against step workload. Experiments show that plugging CPU improves performance of web tier. The work lacks a possibility of CPU capping, therefore spare CPU cycles cannot be used by other VMs. Moreover, applying real workload trace may require improvement of used provisioning algorithm.

VII. Conclusions

In this paper we presented online scaling controller with support of prediction model. We apply vertical scaling using combination of CPU power tuning and CPU hotplugging.

Experiment results show that applying our controller allows to improve CPU usage efficiency with minimum SLA violation rate. Moreover, web server VM provisioned by our controller

achieves higher throughput in comparison to statically provisioned VM.

Additionally we run our controller to provision two prioritized VMs sharing same physical CPUs. Evaluation results show, that our implementation outperforms standard *Xen credit-scheduler* and provides stable response time for time critical application running inside high priority VM. Finally, we discuss advantages of applying vertical scaling and shrinking renting duration from hour granularity to second granularity.

VIII. Future work

We illustrated efficiency of our online controller, but it has some limitations. Currently our controller performs provisioning of only one resource bottleneck - CPU, while computer systems have another types of resources which may affect application performance. In a future research we plan to develop algorithms to determine which resource is bottleneck and has to be extended.

Vertical scaling limited by power of host, while horizontal scaling extends this boundaries. In the future we want to combine vertical and horizontal scaling to implement efficient VM provisioning.

Acknowledgment

This work is supported by Erasmus Mundus Scholarship.

References

[1] Amazon auto scaling service. http://aws.amazon.com/autoscaling/.

[2] Rightscale web site. http://www.rightscale.com.

[3] F. Galán, A. Sampaio, L. Rodero-Merino, I. Loy, V. Gil, and L. M. Vaquero, "Service specification in cloud environments based on extensions to open standards," in *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*, ser. COMSWARE '09. New York, NY, USA: ACM, 2009, pp. 19:1–19:12. [Online]. Available: http://doi.acm.org/10.1145/1621890.1621915

[4] L. Rodero-Merino, L. M. Vaquero, V. Gil, F. Galán, J. Fontán, R. S. Montero, and I. M. Llorente, "From infrastructure delivery to service management in clouds," *Future Gener. Comput. Syst.*, vol. 26, pp. 1226–1240, October 2010. [Online]. Available: http://dx.doi.org/10.1016/j.future.2010.02.013

[5] Z. Hill, J. Li, M. Mao, A. Ruiz-Alvarez, and M. Humphrey, "Early observations on the performance of windows azure," in *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing*, ser. HPDC '10. New York, NY, USA: ACM, 2010, pp. 367–376. [Online]. Available: http://doi.acm.org/10.1145/1851476.1851532

[6] D. Bellenger, J. Bertram, A. Budina, A. Koschel, B. Pfänder, C. Serowy, I. Astrova, S. G. Grivas, and M. Schaaf, "Scaling in cloud environments," in *Proceedings of the 15th WSEAS international conference on Computers*. Stevens Point, Wisconsin, USA: World Scientific and Engineering Academy and Society (WSEAS), 2011, pp. 145–150. [Online]. Available: http://dl.acm.org/citation.cfm?id=2028299.2028329

[7] Z. Ou, A. Y.-J. Hao Zhuang, Jukka K. Nurminen, , and P. Hui, "Exploiting hardware heterogeneity within the same instance type of amazon ec2," in *In Proceeding 4th USENIX Workshop on Hot Topics in Cloud Computing*, ser. HotCloud '12, 2012.

[8] Joyent cloud. http://www.joyentcloud.com/.

[9] C. M. Wesam Dawoud, Ibrahim Takouna, "Elastic virtual machine for fine-grained cloud resource provisioning," in *Proceedings of the 4th International Conference on Recent Trends of Computing, Communication and Information Technologies (ObCom 2011)*, ser. CCIS, no. 0269. Tamil Nadu, India: Springer, 12 2011.

[10] Credit-based cpu scheduler. http://wiki.xen.org.

[11] Z. Gong, X. Gu, and J. Wilkes, "Press: Predictive elastic resource scaling for cloud systems," in *CNSM*. IEEE, 2010, pp. 9–16.

[12] P. A. Dinda and D. R. O'Hallaron, "Host load prediction using linear models," *Cluster Computing*, vol. 3, pp. 265–280, October 2000. [Online]. Available: http://dl.acm.org/citation.cfm?id=592893.592958

[13] C. Amza, E. Cecchet, A. Ch, A. L. Cox, S. Elnikety, R. Gil, J. Marguerite, K. Rajamani, and W. Zwaenepoel, "Bottleneck characterization of dynamic web site benchmarks," Tech. Rep., 2002.

[14] Rubis online auction system. http://rubis.ow2.org/.

[15] The ircache project. http://www.ircache.net/.

[16] E. Kalyvianaki, T. Charalambous, and S. Hand, "Self-adaptive and self-configured cpu resource provisioning for virtualized servers using kalman filters," in *Proceedings of the 6th international conference on Autonomic computing*, ser. ICAC '09. New York, NY, USA: ACM, 2009, pp. 117–126. [Online]. Available: http://doi.acm.org/10.1145/1555228.1555261

[17] A. Dubey, R. Mehrotra, S. Abdelwahed, and A. Tantawi, "Performance modeling of distributed multi-tier enterprise systems," *SIGMETRICS Perform. Eval. Rev.*, vol. 37, no. 2, pp. 9–11, Oct. 2009. [Online]. Available: http://doi.acm.org/10.1145/1639562.1639566

[18] B. Urgaonkar, G. Pacifici, P. Shenoy, M. Spreitzer, and A. Tantawi, "An analytical model for multi-tier internet services and its applications," *SIGMETRICS Perform. Eval. Rev.*, vol. 33, no. 1, pp. 291–302, Jun. 2005. [Online]. Available: http://doi.acm.org/10.1145/1071690.1064252