Towards auto-scaling in the cloud: online resource allocation techniques

Lenar Yazdanov

September 2015

1 Introduction

Cloud computing lowered the barrier of entry to an infinite amount of computing resources. Therefore, nowadays any person in the world can rent computing resources to run an application. Usually the resources are delivered in the form of virtual machines (VMs). In comparison to traditional provisioning techniques that require upfront servers deployment, cloud users can acquire and release resources on-demand. For example, the user can add a VM to the application cluster to cope with an increased workload or terminate idle VM. However, it is not easy to answer the question about *when* to allocate and *how many* resources to allocate.

To identify the right amount of resources to lease the user needs to consider a number of factors: such as application elasticity, workload dynamics, user-defined performance objectives and conversion of the performance objective to resource allocation. For a non-expert cloud user that has limited knowledge about the application and its resource demand pattern it is hard to make optimal scaling decision.

Cloud market offers a variety of resource allocation schemes. The user can choose a VM from the set of predefined templates or specify a VM he needs. Later on, during runtime it is possible to change the application resource capacity by modifying the number of VMs dedicated to the application (*horizontal scaling*) or adapt individual VM resources (*vertical scaling*). The number of possible resource allocation strategies becomes too large to be managed by a human. Therefore, there is a need for automating the process of resource allocation. Auto-scaling services offered by cloud providers simplify the process of acquiring and releasing resources, but leave burden of scaling policy design to the user.

The focus of this thesis is techniques and approaches for online scaling policy discovery. There are two objectives that we target. First, the amount of assigned resources should be enough to meet the application performance goal. Second, the cost of running the application on the cloud should be minimal.

To address the challenges and meet the objectives we present four techniques for online resource allocation. Below presented a short description of our contributions:

1. We design a resource allocation controller that resolves resource conflicts between interactive and batch applications collocated on a host. For cloud providers that support vertical scaling there is a need to utilize residual resources. The presented controller follows the resource demand of the interactive application and allows to utilize residual resources.

2. We design the controller, called Vscaler that performs web application resource provisioning. With the help of reinforcement learning the controller dynamically learns the application performance model at runtime. It adapts the resource allocation to meet the application performance goal and minimize the level of resource over-provisioning.

3. We implemented VscalerLight that extends Vscaler for multi-tier web applications. Applying reinforcement learning (RL) to multi-tier application increases the complexity of the state-space model. To address the issue we analyze the impact of individual VM resources on the application performance and propose to reduce the complexity by creating two separate RL models for RAM and CPU.

4. In our last work we address horizontal scaling of batch applications running in an Elastic MapReduce (EMR) cluster. Scaling the number of compute nodes of the EMR cluster beyond a certain limit, risks 'prolonging' the actual task completion time, because the data nodes and/or the network cannot keep up with the increased demand. ElasitcYARN detects the limit at jobs runtime and adapts the compute part size to minimize the cost of MapReduce job execution.

In the short version of the dissertation we present each of the contributions. Following sections provide a high level description of the applied techniques and present experimental results that highlight the benefits of the presented technique. Finally, we give a brief summary of the work and present possible directions for future research.

2 Vertical scaling for prioritized VMs provisioning

Resource demand of many applications is not static and varies over the time. To achieve high performance of applications users are forced to acquire VMs based on the application peak resource demand. However, peak load resource allocation leads to resource wastage. As a result, users pay for resources that have not been utilized. For cloud users it would be beneficial to have possibility to reconfigure VM at runtime. However, cloud providers that support vertical elasticity face a challenge on how to utilize residual resources on the host.

According to research analysis of datacenters workloads [5, 15, 13, 12] applications running in the cloud can be classified into two groups: latency-sensitive interactive applications and batch applications. The first group consists of web applications such as internet stores, bookkeeping sites and etc. Low latency is an essential requirement for these applications. None of the application users would like to interact with a slowly responding website. For e-commerce applications high latency means potential revenue

loss, because users will eventually move to more responsive web sites. The second group consists of applications running as back-end tasks such as MapReduce jobs. Batch applications do not require real-time responsiveness and can tolerate performance slowdown.

To provide low latency for the interactive application and improve utilization of the host we propose to collocate two classes of applications. We design an online resource allocation controller that performs vertical scaling of collocated VMs. The controller uses VM CPU usage traces to predict future resource demand and triggers scaling actions. It resolves resource conflicts between the applications as follows. If the interactive application's (high priority) resource demand exceeds capacity of the VM, then the controller rents resources from the batch application (low priority) VM. As a result, the interactive application response time does not increase significantly and the batch application makes a progress even during resource contention. If the high priority VM has residual resources, then Vscaler assigns them to the low priority VM.

Results

We evaluate Vscaler against real-world workload traces. For the evaluation we use Apache web server running an e-commerce application and a Hadoop framework that executes MapReduce job. We compare our controller against popular Xen hypervisor with build-in prioritization mechanisms. The Xen scheduler can run in two modes: work-conserving (wc) and non-work-conserving (nwc). In wc-mode each VM is assigned a weight. In this mode sharing (weight) is guaranteed. A host CPU is idle, only if there is no active VM. Therefore, the batch application can get residual CPU cycles from the interactive application. In nwc-mode shares are capped. It means that, in case of two VMs with equal shares, each of the VMs gets 50% CPU, even if second half of CPU is idle.

Figure 1 shows the web server 95% response time of all evaluated configurations. The response time provided by our controller is the most closest to the single VM mode. The response time achieved by the Xen credit scheduler in wc-mode is higher by almost 80 ms. In default mode (non-weighted) CPU-intensive batch application steals CPU cycles from the interactive application and the response time becomes even higher.

In figure 2 we plot the execution time of MapReduce job. The CPU allocation scheme implemented by the Xen credit scheduler in nwc and wc modes provides almost similar execution time. If we apply our controller, then the job runs 2 times longer, than under control of Xen. Vscaler serves the interactive application with high priority. We assume that a user running the batch application acquires compute resources for a lower price and aware of its possible performance slowdown.

By default, Xen credit scheduler uses 30 ms time slice for the CPU assignment. It means that a VCPU of each VM gets 30 ms before being preempted. To provide higher performance for the interactive application in wc-mode we would need to lower the length of the credit scheduler's time slice. It increases the overhead of context switching and reduce effectiveness of a CPU cache [3].

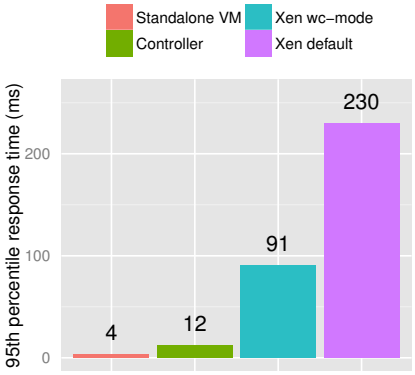

Figure 1: Web server response time

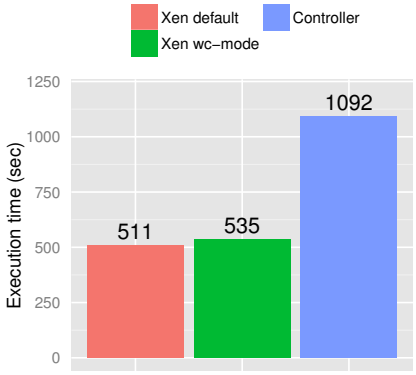

Figure 2: Hadoop execution time

3 Autonomic Virtual Machine Scaling

Recent observations by Agmon Ben-Yehuda et al. [1] of IaaS trends state, that the model of fixed bundles, so called "instance types" will eventually change to flexible bundles. The change is mostly economically driven. For example, the provider offers a VM with 6 CPUs, while the user needs a VM with 5 CPUs. The fixed bundles model does not reflect users economical expectations of pay-as-you-go model. Authors conclude that IaaS providers will eventually shrink billing periods and allow users to build a VM they want to run. The presented trends already exist in the cloud market. Cloud providers, such as CloudSigma, ProfitBricks and GridSpot, offer virtual resources in the form of flexible bundles.

Many web applications have varying resource demand. The model of flexible bundles facilitates more efficient resource provisioning for such applications. Users can dynamically resize VMs based on current resource demand. However, to perform efficient resource provisioning (reduce under-utilization and meet application performance goals) one has to design scaling policy.

Most of public cloud providers offer auto-scaling services at the IaaS level. The services exploit threshold-based scaling approach. The approach tends to focus on scaling at the machine or VM level. But it does not facilitate the definition of higher business function, such as user-specified quality of service (QoS). Using threshold-based scaling, it is hard to convert a VM capacity to the application performance. Moreover, cloud providers shift responsibility of determining a 'good' scaling policy to the non-expert user. We need an approach that allows to discover the scaling policy online.

Reinforcement learning (RL) [9] does not require *a priory knowledge*. It is able to perform online policy learning and adapt to environmental changes. To learn an environment the RL agent takes actions and observes new states. For every action it obtains a reward. The time it takes to activate every action and visit all states of the environment depends on the size of state-action space. In resource allocation problem the state

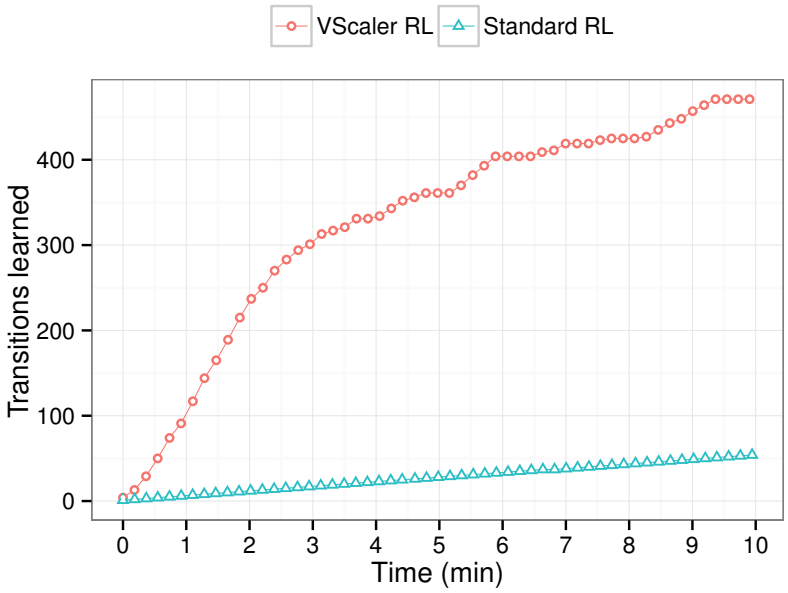

Figure 3: Transitions learned

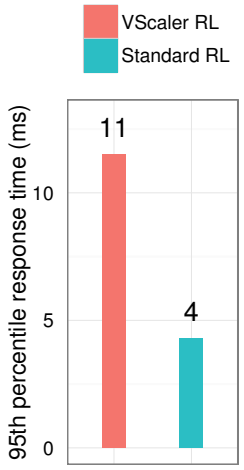

Figure 4: Application 95% response time

determines the amount of resources assigned to a VM and the action defines the amount of resources to acquire or release. It means that the agent has to visit a significant fraction of the environment, before it can take optimal decisions.

Our solution is based on the idea that there is more to learn from a single transition. Every time when action is taken, the agent observes amount of resources consumed by the application and obtains a reward for the taken action. However, if there are alternative states where VM capacity is higher than observed resource demand, then we can update transitions that connect initial state with the alternative states.

Results

To show the learning speed-up provided by VScaler we evaluated two Q-learning algorithms. The first algorithm uses our approach. The second one is a standard Q-learning approach, where the agent after each observation updates only one state-action pair. In both approaches the agent learns the environment using a standard policy. It means that with a probability of $\varepsilon = 0.05$ the agent takes a random action. In the exploitation phase the agent takes an action that gives the higher utility. For the evaluation we use the RUBiS benchmark that simulates users behavior of online auction. In the experiment we run the constant workload.

In figure 3 we show the number of transitions learned. During the first 3 minutes VScaler RL learns 300 transitions, while Standard RL learns significantly less transitions. VScaler RL has a higher learning rate, but the most important part is the quality of

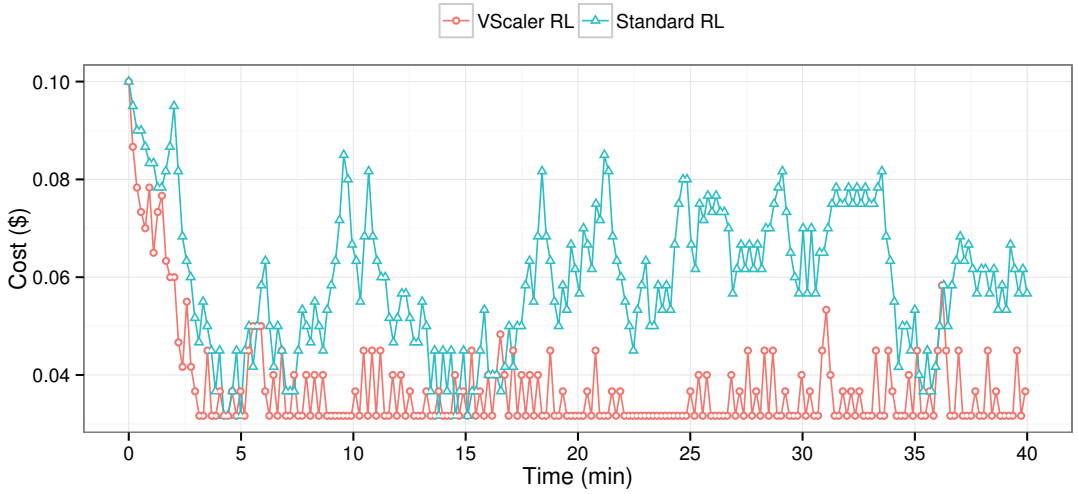

Figure 5: VM costs: Standard RL vs VScaler RL

resource allocation policy obtained by the approaches. Figure 4 presents the response time delivered by the web server under the control of the evaluated learning models. We ran the experiment for 40 minutes. For the experiment we set desired response time to be below 20 ms. Both learning approaches keep 95% of response time below 20 ms.

The standard RL model has only information about actually visited transitions, while the VScaler RL in addition to visited transitions knows about the impact of alternative transitions. The additional information allows the VScaler RL to quickly converge to the state with the minimal amount of required resources. In figure 5 we present the cost of resources in each state. We consider a flexible bundles resource model that allows to dynamically modify individual resources assigned to a VM. For the experiment we took prices from CloudSigma. The figure shows that the VScaler RL only needs 3 minutes to find the optimal VM size.

The VScaler RL quickly adapts to the workload, while the Standard RL needs more time to learn the environment. One has to notice that both approaches do not violate the SLA, but the VScaler RL in comparison to the Standard RL achieves the performance goal for the lower cost.

4 Autonomic Multi-tier application Scaling

The second well-known problem of the reinforcement learning approach is the curse of dimensionality. The state-space dramatically grows with increased number of parameters that describe the model of the application. To apply RL to multi-tier web application provisioning we need to include each tier VM parameters into the model. It is common to reduce the state-space and the actions-space to address the issue. However, it leads to coarse-granular resource allocation.

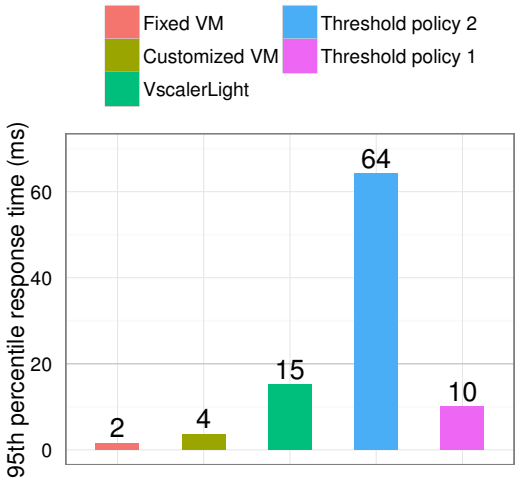

Figure 6: 95% response time

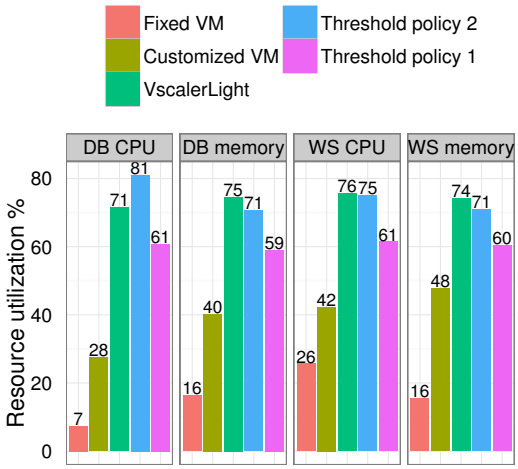

Figure 7: Each tier VM resource usage

We analyzed the impact of individual VM resources (CPU and RAM) on performance of multi-tier web-application. CPU and RAM belong to different groups of resources. CPU is a *compressible* resource (as well as network and disk I/O bandwidth) and memory is *non-compressible* resource (as well as disk space). To smoothly regulate the response time of the application we can throttle virtual CPU power assigned to a VM. In contrast, memory cannot be used to control the response time. The response time sharply increases if memory utilization reaches certain (about 90%) threshold. The reason is swapping process that triggers when we reclaim memory utilized by the application.

To meet user-specified performance objective and control resource allocation we created two separate models: CPU and RAM. CPU model regulates the application response time and RAM model changes memory allocation to avoid swapping. Moreover, we addressed cluster wide correlation effects that can cause shift of resource bottlenecks. To orchestrate the resource allocation across the tiers we added workload description parameter to each of the RL models.

Results

In our evaluation we compare VscalerLight against threshold-based scaling policies and two static allocation schemes. Cloud user can use an auto-scaling service offered by a provider to perform dynamic resource assignment. However, it is necessary to find optimal threshold values for the application. We empirically define two policies. First policy tries to minimize the level of over-provisioning. The second one aims to minimize SLA violation events. For static allocation schemes we assume that the user knows expected resource demand and assigns VM resources according to the peak demand. For the first scheme the user takes fixed size VM that is equal to Amazon EC2 small

instance [6]. For the second one the user specifies VM size. As multi-tier web application we used RUBiS benchmark that consists of web and database servers. To emulate real user behavior we took 6 hours long workload trace from World Cup 98.

Figures 6 and 7 show the performance and resource usage provided by the evaluated schemes. The policy 2 violates SLA, while the rest schemes keep the application response time below specified value (20 ms). The figure 7 presents the efficiency of each allocation scheme. The higher the value, the better resource usage efficiency. Fixed size VM allocation leads high resource wastage. Resource utilization of both VMs is below 26%. It means that in the cloud market the user overpays for about 74% of allocated resources. If user customizes VM capacity, then the utilization can be improved by factor of 2 in comparison to fixed size VM. However, one needs to know workload upfront. Dynamic resource scaling improves resource utilization even further. In all dynamic schemes the utilization is above 59%. However, only two of them (VscalerLight and policy 1) meet user-specified performance objective. But VscalerLight achieves 10% higher utilization in comparison with threshold policy 1 and does not require tuning of the thresholds upfront.

5 I/O aware elastic mapreduce cluster scaling

To allow users run data processing jobs, public cloud providers, such as Amazon, offer Elastic MapReduce (EMR), a web service for MapReduce applications. In contrast to traditional data-processing clusters, EMR has dedicated nodes for data and computation. The key characteristic of EMR is elasticity. The compute nodes of EMR cluster can be scaled out to speed up a job. Increasing the number of compute nodes increases the network traffic as all compute nodes need to access to the whole data set during the map and the reduce phases. Scaling the number of compute nodes beyond a certain limit, risks 'prolonging' the actual task completion time, because the data nodes and/or the network cannot keep up with the increased demand. As a result, it increases the resource usage time that reflects the total cost of running MapReduce job in the cloud. To minimize the cost of the job execution in the cloud it is important to scale compute nodes with respect to bandwidth delivered by the data nodes and the network.

We also observe a trend towards a cross-cloud data processing [4, 7, 8] and the use of Micro-clouds [14, 11]. Running data processing in a cross-cloud fashion incurs WAN data transfer. We have to make sure that jobs running in such environment do not suffer due the network saturation.

The pay-as-you-go model and the elastic nature of the platform allow the user to change the size of data processing cluster almost instantaneously. Agmon Ben-Yehuda et al. [1] observe that IaaS providers shift from hour range to seconds range billing cycles. The shorter billing cycles allow users to remove exceeding virtual resources at any point of time, without waiting for the end of an hour billing cycle [10, 2]. However, to achieve the cost efficiency the user has to scale the compute nodes with respect to the data part capacity and the available network throughput. Average user does not have upfront knowledge about performance delivered by these resources and MapReduce job's

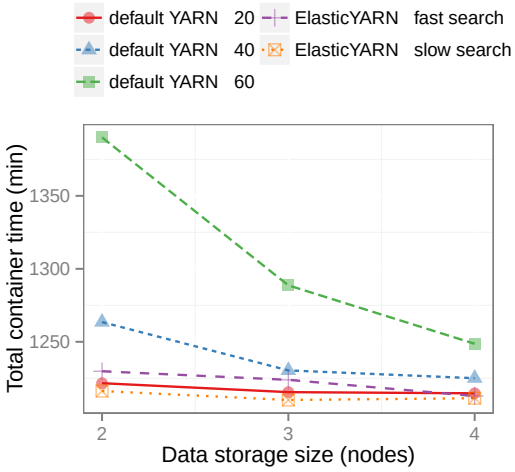

Figure 8: Inter-cloud deployment

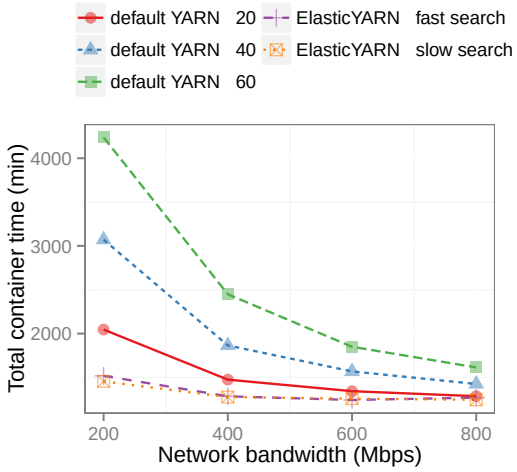

Figure 9: Cross-cloud deployment

bandwidth requirements. Hence, there is a need for a system that can resize a compute cluster of EMR to minimize the cost of job execution in the cloud.

We analyzed the model of MapReduce execution. The task of each phase (map or reduce) performs two types of I/O. In map phase it reads the data over the network and in the spill and the merge stages it writes data to the disk. In reduce phase the task stores the data that was fetched during the shuffle stage on the disk and then sends the output of a reduce function to the data nodes. If there are no bottlenecks then CPU utilization is the same during network I/O operations and disk write activity. To identify the presence of the bottlenecks and calculate the maximal number of the compute nodes that can run without hitting the limit, we compare the CPU usage of the two I/O operations.

Results

To demonstrate the effectiveness of the presented approach, we have developed a system called ElasticYARN. ElasticYARN runs on top of YARN framework that executes tasks of MapReduce job in containers. The size of the compute cluster depends on the number of containers assigned to a job. The limit and the job bandwidth requirements are not known upfront. Therefore, ElasticYARN gradually scales the number containers in the wave (tasks running in parallel compose a wave). As soon as the limit has been detected, the system assigns the optimal number of containers. We evaluate two configurations of ElasticYARN that define how to increase the wave size if the limit has not been detected. In the first configuration the system doubles the number of containers in the wave; in the second configuration it increases the wave size by a fixed value (5% of phase size). We ran four different jobs to evaluate the system: Sort, Wordcount, Hive join and Hive aggregate.

In the evaluation, we consider two scenarios: inter-cloud deployment and cross-cloud deployment. In the inter-cloud deployment we vary the data storage capacity. The network capacity is varied in the cross-cloud deployment. Figures 8 and 9 show the total container time that was spent to complete the jobs. In both scenarios the total container time consumed by ElasticYARN to complete the jobs is minimal comparing to the default YARN. The slow search mode provides lower the total container time in comparison to the fast search mode, because the number of tasks in the wave hitting the limit is smaller. The total container time of the jobs managed by ElasticYARN does not vary significantly across all configurations.

6 Conclusion

Economic interests of cloud users already resulted in changes on the cloud market. There are public cloud providers that addressed the users expectations and shifted to a flexible VM model. The users are free to specify a VM size they need and can change it during runtime. Recently, the providers start to move from hour billing cycles to second billing cycles. However, there is not much improvement regarding to auto-scaling services. The focus of this thesis is to make one step forward to address the cloud market changes and propose auto-scaling techniques. In this work we design controllers that automatically perform scaling decisions to meet the application performance objectives and minimize the virtual resources usage costs.

Our contributions are summarized as follows:

1. To enable vertical elasticity we propose a priority aware controller. In future we would like to extend the controller to provide job completion time guarantees for the low priority batch applications.

2. It is hard to meet the user-specified performance objective by provisioning the application based on current resource demand. To enable QoS aware resource provisioning, we designed online resource allocation controller for single VM web applications. Vscaler is model-free controller that learns scaling policy in online fashion.

3. Many web-applications consist of multiple tiers. To control resource allocation across the tiers we implemented VscalerLight. The controller maximizes the resource utilization of the application tiers and meets the user-specified performance objective. Vertical elasticity is naturally limited by capacity of the host. To deal with larger workloads we have to exploit horizontal scaling. In the future, we want to combine vertical and horizontal scaling to serve larger workloads.

4. In the last work, we address the resource allocation of the batch applications running in EMR cluster. We implemented ElasticYARN, a system that detects MapReduce job traffic demand and rescales EMR cluster to minimize the cost of the job execution in cloud environment. Further, we want extend the YARN scheduler to make it I/O aware.

References

[1] Orna Agmon Ben-Yehuda, Muli Ben-Yehuda, Assaf Schuster, and Dan Tsafrir. The rise of raas: The resource-as-a-service cloud. *Commun. ACM*, 57(7):76–84, July 2014. ISSN 0001-0782. doi: 10.1145/2627422. URL http://doi.acm.org/10.1145/2627422.

[2] Emiliano Casalicchio and Luca Silvestri. Mechanisms for {SLA} provisioning in cloud-based service providers. *Computer Networks*, 57(3):795 – 810, 2013. ISSN 1389-1286. doi: http://dx.doi.org/10.1016/j.comnet.2012.10.020. URL http://www.sciencedirect.com/science/article/pii/S1389128612003763.

[3] Ludmila Cherkasova, Diwaker Gupta, and Amin Vahdat. Comparison of the three cpu schedulers in xen. *SIGMETRICS Perform. Eval. Rev.*, 35(2):42–51, September 2007. ISSN 0163-5999. doi: 10.1145/1330555.1330556. URL http://doi.acm.org/10.1145/1330555.1330556.

[4] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google’s globally distributed database. *ACM Trans. Comput. Syst.*, 31(3):8:1–8:22, August 2013. ISSN 0734-2071. doi: 10.1145/2491245. URL http://doi.acm.org/10.1145/2491245.

[5] Christina Delimitrou and Christos Kozyrakis. Quasar: Resource-efficient and qos-aware cluster management. In *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '14, pages 127–144, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2305-5. doi: 10.1145/2541940.2541941. URL http://doi.acm.org/10.1145/2541940.2541941.

[6] ECU. Amazon ec2 compute unit. URL https://huanliu.wordpress.com/2010/06/14/amazons-physical-hardware-and-ec2-compute-unit.

[7] Andy Edmonds, Thijs Metsch, Dana Petcu, Erik Elmroth, Jamie Marshall, and Plamen Ganchosov. Fluidcloud: An open framework for relocation of cloud services. In *Presented as part of the 5th USENIX Workshop on Hot Topics in Cloud Computing*, Berkeley, CA, 2013. USENIX. URL https://www.usenix.org/conference/hotcloud13/workshop-program/presentations/Edmonds.

[8] Anca Iordache, Christine Morin, Nikos Parlavantzas, Eugen Feller, and Pierre Riteau. Resilin: Elastic mapreduce over multiple clouds. *2014 14th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing*, 0:261–268, 2013. doi: http://doi.ieeecomputersociety.org/10.1109/CCGrid.2013.48.

[9] Leslie Pack Kaelbling, Michael L. Littman, and Andrew W. Moore. Reinforcement learning: A survey. *J. Artif. Int. Res.*, 4(1):237–285, May 1996. ISSN 1076-9757. URL http://dl.acm.org/citation.cfm?id=1622737.1622748.

[10] J Kupferman, J Silverman, P Jara, and J Browne. Scaling into the cloud,(2009). *URL http://cs. ucsb. edu/˜ jkupferman/docs/ScalingIntoTheClouds. pdf (available online accessed 29.01. 12)*.

[11] Jie Liu, Michel Goraczko, Sean James, Christian Belady, Jiakang Lu, and Kamin Whitehouse. The data furnace: Heating up with cloud computing. In *Proceedings of the 3rd USENIX Conference on Hot Topics in Cloud Computing*, HotCloud'11, pages 15–15, Berkeley, CA, USA, 2011. USENIX Association. URL http://dl.acm.org/citation.cfm?id=2170444.2170459.

[12] Charles Reiss, Alexey Tumanov, Gregory R. Ganger, Randy H. Katz, and Michael A. Kozuch. Heterogeneity and dynamicity of clouds at scale: Google trace analysis. In *Proceedings of the Third ACM Symposium on Cloud Computing*, SoCC '12, pages 7:1–7:13, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-1761-0. doi: 10.1145/2391229.2391236. URL http://doi.acm.org/10.1145/2391229.2391236.

[13] Malte Schwarzkopf, Andy Konwinski, Michael Abd-El-Malek, and John Wilkes. Omega: Flexible, scalable schedulers for large compute clusters. In *Proceedings of the 8th ACM European Conference on Computer Systems*, EuroSys '13, pages 351–364, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1994-2. doi: 10.1145/2465351.2465386. URL http://doi.acm.org/10.1145/2465351.2465386.

[14] Frezewd Lemma Tena, Thomas Knauth, and Christof Fetzer. Powercass: Energy efficient, consistent hashing based storage for micro clouds based infrastructure. In *Proceedings of the 2014 IEEE International Conference on Cloud Computing*, CLOUD '14, pages 48–55, Washington, DC, USA, 2014. IEEE Computer Society. ISBN 978-1-4799-5063-8. doi: 10.1109/CLOUD.2014.17. URL http://dx.doi.org/10.1109/CLOUD.2014.17.

[15] Abhishek Verma, Luis Pedrosa, Madhukar R. Korupolu, David Oppenheimer, Eric Tune, and John Wilkes. Large-scale cluster management at Google with Borg. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, Bordeaux, France, 2015.