Orchestrating Resource Allocation for Interactive vs. Batch Services using a Hybrid Controller

Bilal Ahmad*, Anis Yazidi*, Hårek Haugerud* and Soodeh Farokhi†
*Oslo and Akershus University College (HiOA), Norway
†C2RO - Collaborative Cloud Robotics, 780 Avenue Brewster, Montreal, Canada

***Abstract*—Cloud service providers are trying to reduce their operating costs while offering their services with a higher quality via resorting to the concept of elasticity. However, the vast majority of related work focuses solely on guaranteeing the quality of service (QoS) of interactive applications such as Web services. Nevertheless, a broad range of applications have different QoS constraints that do not fall under the same class of latency-critical applications. For instance, batch processing possesses QoS requirements that are latency-tolerant and usually defined in terms of job progress. In this sense, a possible manner to quantify the performance of a batch processing application is to estimate its job progress so that to determine if future deadlines can be met. The novelty of this work is two-fold. First, we propose a hybrid controller coordinating resource allocation between interactive and batch applications running at the same infrastructure. The intuition is to deploy a controller for the interactive application at a faster time-scale than the batch application. Second, we bridge the gap between vertical and horizontal scaling under the same framework. In this perspective, vertical scaling is used for small fluctuations in the load, while horizontal scaling handles larger load changes. Comprehensive experimental results demonstrate the feasibility of our approach and its efficiency in ensuring a high CPU utilization across all experiments consisting of 83.70% for the Web service and 89.51% for the batch service, while meeting the respective QoS requirements of both services.**

***Index Terms*—Resource Allocation, Autonomous Computing, Control Theory, Web Service, Batch Application, Quality of Service (QoS), Response Time, Job Progress**

I. Introduction

Cloud computing is an emerging technology and is becoming more popular, due to advantages such as elasticity and infinite computing resources. Companies are increasingly taking advantage of the benefits and moving their infrastructure to the cloud to reduce the operational cost. According to a study performed by Natural Resources Defense Council in 2014 [1], the main issue for energy saving is under utilization of data centers. In addition, data centers have the fastest growing consumption of electricity in United States [1]. It is estimated that Google Web search servers often have an idleness of 30% over a 24 hour period [2]. It means, if we envisage a cluster of 20,000 servers, the capacity of 6,000 servers would be wasted. Consequently, maximizing server consolidation would cut unnecessary energy and operation costs and increase return on investment.

From a top-down perspective, we can argue that cloud computing has entangled with the concept of elasticity and virtualization to address the under-utilization issue of grid computing or cluster computing. In a nutshell, elasticity is achieved via horizontal scaling and/or vertical scaling. The concept of horizontal scaling is the de facto standard because of its simplicity, as it does not require any extra support from the hypervisor [3]. Horizontal scaling relies on increasing the capacity by connecting multiple hardware or VMs and orchestrating their work as a single entity. While vertical scaling consists of adding more resources to a single node in a system. Furthermore, horizontal elasticity is coarse-grained, which means that a CPU core can dynamically be leased to a VM for a certain amount of time. While vertical elasticity is fine-grained, fractions of a CPU core can be leased for as short as a few seconds [4]. It is worth-mentioning that despite the clear advantages of vertical elasticity, there is only a limited amount of research focusing on this subject mainly due to its increased complexity [5], [6], [7].

Applications hosted on VMs have different demands when it comes to the quality of service (QoS). Interactive applications are latency-critical and sensitive to unpredictable spikes in user access, even a small amount of interference can cause significant QoS degradation. On the other hand, batch applications are less sensitive to various instantaneous resource disturbance conditions. One of the challenges of cloud infrastructure providers is that they do not know what kind of applications are running on their infrastructure so they cannot effectively adjust the resource allocation to achieve a preferred QoS.

The aim of this paper is to propose and design an autonomic resource allocation controller using *control theoretical approaches* to manage QoS of heterogeneous application types. Application level metrics of interactive and batch applications will be used as an indicator of QoS. The **contributions** of this paper are twofold:

- It devises an autonomic resource allocation controller using control theoretical approaches to increase the server utilization hosting both interactive and batch services, while allocating sufficient resources to ensure the desired QoS of both types of applications.
- It bridges the gap between vertical and horizontal scaling schemes in a designed hybrid controller. The vertical

scaling is used to tackle small fluctuations in the input workload, while the horizontal scaling handles larger workload changes.

The remaining of this paper are as follows. In Section II, the state-of-the-art on resource allocation in cloud computing is reviewed. Section III presents the design of the proposed hybrid controller. In Section IV, the experimental evaluation results are discussed. Finally, Section V concludes the paper and envisions the future work.

II. Related research

The concept of *self-adaptive cloud environments* is not new, it covers a broad area of research fields, where there is still ongoing extensive research. Because of the increased use of *cloud computing* [8], cloud service providers are encountering new challenges to ensure SLA and QoS requirements. There is a significant research on achieving increased efficiency and better resource management. In [7], the authors explore vertical elasticity features in cloud computing environments. The focus in the study was completely on scaling memory using control theory. The decision maker compares the desired and actual response time (RT) of the application and adjusts the memory allocation accordingly. The results of the experiments show a significant increase of the memory efficiency by at least 47%. Since applications in most cases are dependent of a combination of memory and CPU, a coordination between the resources is essential for efficient resource utilization. The study reported in [9] describes the novelty of the research by using a fuzzy control approach as a resource coordinator between memory and CPU controller. The study shows that without having any coordination between the memory and CPU controller, the VM is in most cases under- or over-provisioned with resources. The control logic is based on Fuzzy rules which include; RT, utilization of CPU and memory as a performance vector. Comparing the results of using fuzzy controller and non-fuzzy controller shows that without having any coordination between the controllers most of the times one of the controller over-provisions resources. By coordinating the controllers, the right amount of resources is allocated to meet the desired response time of the application. In [10], Farokhi addressed the problem of controlling the trade-off between QoS and cost. This paper investigates models, algorithms and mechanisms to handle these two perspectives:

- The first approach is concerned with the cloud providers point of view to offer a distributed infrastructure placement of virtual machines. In this approach the *Bayesian* network model is used to perform decision making.
- In the second approach, the author looks at the trade-off between QoS and cost from the cloud customers point of view. The concept of Fuzzy controller is used to coordinate the resource controllers to meet the performance in a cost-effective way.

The results from the study shows that with the trade-off between QoS and cost for the cloud provider, the proposed approach managed to decrease the energy cost in the infrastructure by up to 69% in comparison to the first state-of-the-art baseline algorithm, and 45% in comparison to the second algorithm. The second approach seeks a trade-off between QoS and cost for the cloud customers. Several experiments were conducted with real-world workload traces. They managed to efficiently save at least 47% memory usage while keeping the desired performance level. By virtue of the coordination between resources with the Fuzzy controller, the experiments results showed a reduction of the memory usage by up to 60% in one of the scenarios and up to 56% less CPU usage in another one, compared to not having any coordination between the controllers.

The main focus of [11] lies on increasing resource efficiency by reusing resources of underutilized servers in a production environment. The authors present a feedback based controller, named *Heracles*, which coordinate resources between best-efforts tasks and latency-critical services (LC). The desired goal is to keep the service level objectives (SLO), and a small interference could cause SLO violations for the latency-critical service. The focus is to maintain and guarantee that the LC service receives enough amount of shared-resources, memory, CPU and network I/O. Results from the work showed that *Heracles* managed to increase average utilization of 90% across different scenarios without any SLO violation for LC tasks in a production environment.

Resource provisioning is typically coarse-grained, this means that CPU cores are typically leased for periods as long as one hour. Vertical scaling has improved resource efficiency, resources can be provisioned for as least a few seconds. An empirical study uses the mean of response time to measure QoS of popular cloud applications[4]. The interesting points made in the study is that response time is not in a linear relationship with capacity. By presenting a model called *Queue Length Model*, the relationship is presented as $q = \lambda \cdot R$ where q is is the average queue length, λ is the arrival rate and R is the response time. The second model is called the *Inverse Model*, where the relationship between an application's mean response time R and capacity allocated is represented as $R = \beta/c$. The parameter β is the model parameter and, as in the *queue* model, earlier measurements of capacity and response time is used to calculate β, c is the capacity and R is the response time. The results showed that both models described above managed to predict the needed capacity. In the scenario in which the target response time is low, the *Inverse Model* was more stable than *Queue Length Model*.

Applications in cloud environments are often subject to varying workloads. A study performed by researchers from VMware and University of Würzburg [12], developed a solution for proactive scaling of memory on virtualized applications. The study used statistical forecasting to predict future workloads and scale precisely based on the needed

resources. By using real-world traces to create real scenarios, and comparing both a reactive and proactive controller, the researchers managed to show that performance increased with more than 80% using a proactive controller.

The aim of the study reported in [13] was to develop a controller to perform elastic provisioning of resources to prioritized VMs and avoid SLA-violations. The paper also evaluates the benefits of performing vertical scaling of prioritized VMs. They use real-world workload traces from WorldCup 98 with the Web application RUBis online auction benchmark. CPU scaling was performed with CPU cap by using *Xen credit-scheduler* to adjust the resources. The results from the paper show improvement in CPU usage efficiency without having any major SLA violations. The developed controller achieved better throughout in comparison to a statistical provisioned VM. In addition, the approach yielded a stable low response time for the latency-critical application running on the prioritized VM.

III. Approach

In this section, we give insights into the design of our hybrid controller for coordinating resource allocation between two types of applications: Web service and batch application. The design of our hybrid controller is driven by two key observations:

- A Web service has generally real-time requirements and therefore the controller should able to perform fast elasticity decisions.
- On the other hand, the batch service is less sensitive to resource starvation over short intervals and can make up for execution delay in subsequent intervals. Therefore, the resource allocation to be a batch application can be "uneven" over time as long as the batch is executed within its predefined execution deadline.

A. Controller models

At this juncture, we propose to integrate two types of controller models, a *performance-based* and a *capacity-based* controller. Furthermore, the reason for choosing a combination of the two controller models as foundation for the prototype will also be described in more details.

The **Capacity-based** controller is built upon the concept of allocating resources based on the level of utilization. Capacity-based vertical scaling has been widely adopted by cloud providers because of its simplicity. Utilization of resources is used to estimate the required resources in interactive applications. This does not give any indication on the QoS of the applications, and can in many cases lead to over-provisioning of resources. It is hard to determine what combination of resources an application needs to reduce the chances of violating the SLA, but application level metrics may give a better understanding of whether the application is suffering or not.

The **Performance-based** controller puts emphasis on the QoS rather than on the utilization of resources to perform decision making. The performance is gathered from the application level metrics, such as response time and the metrics give an indication about the latency of the application. The controller has defined levels of acceptable and non-acceptable values and those are used when performing decision making. There are some few research studies that use performance-based controllers, and the results show that the controller manages to increase resource efficiency. Figure 1 illustrates the architecture of the performance- and capacity-based controller. The model is built upon concepts from *control theory*. The red-colored lines illustrate the capacity-based controller. The controller is fed with the desired capacity, and then performs collection of utilization metrics, which then are compared to the desired utilization. The black-colored line illustrates the decisions which can be to either add or remove single or multiple resources to meet the desired utilization. However, if the utilization meets the desired capacity - nothing is done. For the performance-based controller, the blue-colored dashed lines illustrate the model. The desired performance of the application is fed to the controller, then the performance is measured, and the same decisions as explained above are made. However, since choosing a capacity- or performance-based controller does not satisfy the defined criteria, a hybrid version is needed, which consists of a combination of the models. The hybrid version consists of first using the performance-based model to only measure the performance in relation to the desired capacity, and then the capacity-model is used if the performance does not meet the SLA-requirements.

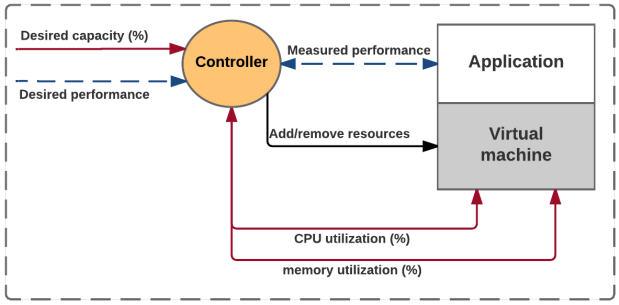

Fig. 1: Capacity- and performance-based controller

B. Decision model

Control theory is used as a foundation for the decision making. The control feedback loop in Figure 2 is based on a feedback control loop.

The desired QoS can be defined as rt_k, and the measured QoS as rt_i. The control error (e_i) is the difference between these two

values in each interval. $Umem_i$ and $Ucpu_i$ are equivalent to utilization of memory and CPU, respectively. While mem_i and cpu_i, is the amount of CPU or memory added. The workload is observed as disturbance, and since the controller has no control over the workload, it adjusts the resources in order to meet the desired QoS.

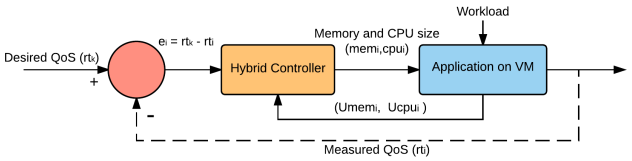

Fig. 2: The feedback control loop for the hybrid controller

This model is adopted both for the interactive application and the batch application, with Response time and frames per second as metrics. As mentioned in the previous section, the hybrid controller utilizes the performance- and capacity-based controller models.

C. Controller metrics

The controller metrics used in this paper can be divided into response time and frames per second. These two SLA-parameters are defined in table I and II, respectively.

Defining how fast the application should respond is not an easy task, since there are not any specific industry standards. However, based on earlier research on human reaction [14]:

- 0.1 second is the limit for the user to feel that the system is reacting instantaneously.
- 1.0 second is the limit for the user to notice the delay.

Our SLA policy for the Web service aspires to keep the average response time to be within the interval of 100 and 500 ms. Each time the response time exceeds 500 ms it is recorded as a SLA-violation. When performing the experiments, the violations of SLA will be monitored. If the average response time drops below 100 ms, this means that more resources than necessary is used and that the allocated resources needs to be reduced.

The batch job's SLA policy has a lower priority compared to the Web server. The desired average frames per second is defined to be within an interval of 15 and 20. There is a limit of max FPS set to 23 FPS. However if the average frames per second drops below 15 there is not any violation. There will be a need for increased resources to make up for the delay in encoded frames. Therefore having an average frames per second which exceeds 20 for some time is not critical. The target QoS measure for the batch service is finishing the job approximately within a time-period of 25 minutes. In other words, the batch job is a task that has a prefixed deadline for being finished. The informed reader observers that our controller operates here at two different time-scales, one slow time-scale and one fast time-scale. In this sense, a fast time -scale control loop is dedicated to the interactive service (Web service), while the slow time-scale control loop is dedicated to the batch computing.

TABLE I: SLA: Web service

Average response time	SLA
Fast	<100 ms
Medium	100-500 ms
Slow	>500 ms

TABLE II: SLA: Batch-job

Average frames per second	SLA
Fast	>20 FPS
Medium	15-20 FPS
Slow	<15 FPS

The two resource metrics that are also taken into consideration by the controller is utilization of CPU and memory, as illustrated in Figure 2. To reduce the chances of either over- or under-provisioning, the controller has defined a level containing minimum- and maximum resources, this is illustrated in table III.

The controller always has the state of the VM monitored, containing the usage of CPU and memory. Using those metrics, when performing vertical down-scaling the minimum resources is defined as the memory used plus a buffer of 512 MB. When scaling down there will then never be an issue that used memory is removed causing memory segmentation faults. The VM will always have a buffer to grow into when needed. In addition, if there is no load or the load is manageable with a single vCPU, then that will be the least possible amount of allocated vCPU.

When performing vertical up-scaling, resources is added when the utilization reaches 80% of the allocated resources, e.g. if a VM has 5 vCPUs and the CPU usage is above 400%, a new vCPU is added. With horizontal up-scaling, the same concept is used, however it is based on the total usage of the server. If the total usage exceeds 80% of available resources, horizontal up-scaling is performed to distribute the load among the servers.

TABLE III: Utilization of resources

Resources	Minimum resources	Maximum resources
Memory	UsedMemory + 512 MB	80% of available resources
CPU	1 vCPU	80% of available resources

The decision logic for some of the functionality is illustrated in Figure 3, starting by measuring the response time and making a decision based on that. Next it checks if the maximum resources of the PM is reached, if not, vertical scaling is performed based on utilization of the VM. However, if the batch VM is not running with the minimum defined resources,

resources are stolen for an amount of time to satisfy QoS-requirements of the interactive application. However, if there are no available resources left, horizontal scaling is performed by booting up a new Web server on the second PM and distributing the traffic between the two Web servers. This is done until the simulated traffic comes back to a level where one Web server is able to handle the traffic load, then down-scaling is performed.

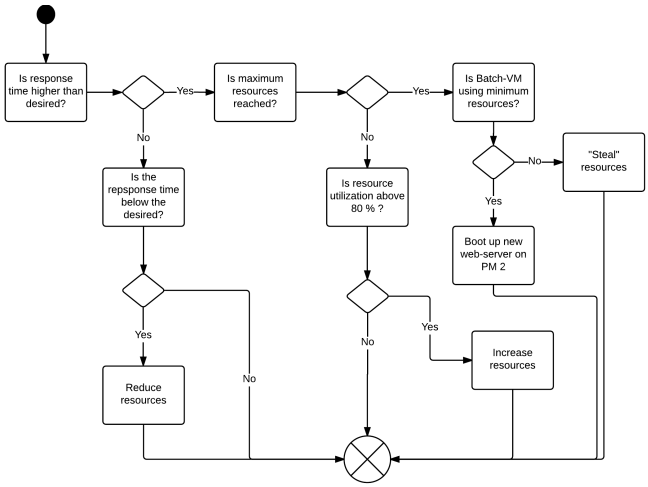

Fig. 3: An activity diagram that summarizes the decision logic determining whether or not to scale based on QoS-requirements.

IV. Implementation and Experimental Results

A. Experimental setup

The physical equipment where the experiments was conducted consists of two Dell PowerEdge R610 physical machines (PMs). Having access to the physical hardware simplifies the control of the resources and how they are allocated. Both of the servers have the same specifications and are running the operating system Ubuntu 12.04.5 LTS. Xen was the only Hypervisor with support for all of the needed features, especially CPU hot-unplugging, and therefore chosen. The PMs have the latest available Xen version 4.1.6.1 configured and installed. Table IV illustrates the specifications of PMs.

TABLE IV: PM specifications

	2xR610
CPU	2xQuad-core Xeon E5530 2.40 GHz
Memory	24 GB (1066 MHz)
Disk	2x146 GB (146 GB in RAID 1)
Network	8xEthernet ports

The main resources which will be used for vertical scaling are CPU and memory. There are in total 16 vCPUs with hyper-threading enabled and 24 GB of memory. These resources will be the limitations when performing vertical scaling.

1) Experimental overview: The infrastructure, as illustrated, can be divided into three components: client, control and server side.

The **Client** side is where workload patterns are fed into the *Loader* which then simulates the traffic by sending HTTP requests and meanwhile measures the response time of the sent requests.

The **Control** side is where the traffic arrives and is further distributed to the Web servers. The *controller* runs at a specified control interval and collects performance metrics from the interactive and batch applications. The control interval for the interactive service has finer granularity than the batch service. The reason is that the Web service has real-time requirements and is more prone to instantaneous changes in the traffic load. On the other hand, the batch service is less sensitive to resource scariness over short intervals and can make up for execution delay in subsequent intervals.

Based on the metrics, a decision is made to either increase or decrease resources through the Xen API. However, the actions are performed if the utilization is above or less than 80% of the available resources. The utilization of resources are collected either directly from the VMs or from the Hypervisor. VM_1 is the *Dom0* and provisioned with sufficient resources to avoid being a bottleneck.

The **Server** side is where the applications are running, divided on two PMs. Except the database VM, all of the other VMs have elastic resources which are adjusted by the *controller* in run-time. The second Web server is booted in VM_5 in the second PM. Both of the RUBBoS Web applications queries the RUBBoS database for each GET request made by Loader.

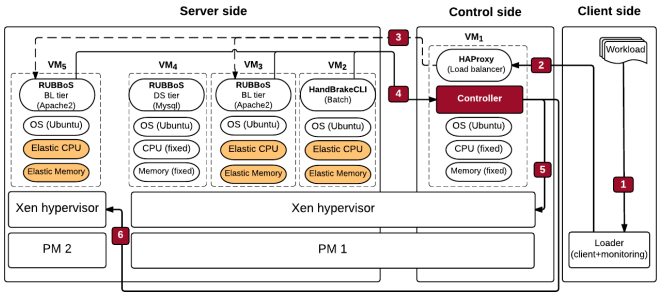

Fig. 4: Experimental overview

HAProxy was configured to balance the Web traffic load between the available Web servers.

For batch processing *HandBrakeCLI* was configured on a separate VM. A video file of 3.1 GB was loaded into the VM and a process of converting the file from *.mp4* to *.mkv* was launched in the experiments. *HandBrakeCLI* is a CPU intensive tool and is able to perform multi-processing with all of the available CPU cores.

2) Workload patterns: Two types of workload patterns were simulated during the experiments, *spiky*- and *trend*-based traffic.

The *spiky* workload pattern, illustrated in Figure 5 has two variables which are used as metrics, the number of clients and requests. The simulation of the traffic consists of sudden spikes in the number of requests: around 20 most of the time and suddenly increasing up to 130 after one, three and and four minutes. There are also some few smaller spikes after the first large spike. The number of clients increases linearly from zero and up to 120 simultaneous clients. The simulated traffic lasts for five minutes.

The *trend* workload pattern, illustrated in Figure 6, is a traffic pattern with a linear increase in the number of clients, from zero and up to 2900 over ten minutes. The number of requests exhibits a stable increase until six minutes and then stabilizes around 800 000 requests.

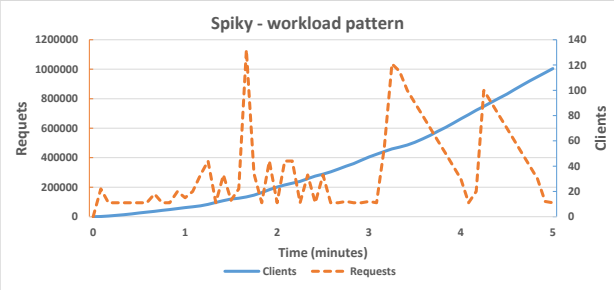

Fig. 5: Spiky workload pattern

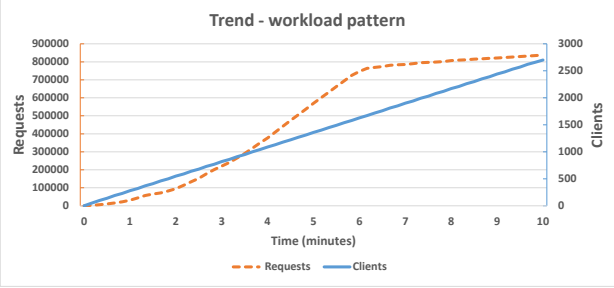

Fig. 6: Workload trend pattern

These two traffic patterns were simulated during the experiments to analyse the behaviour of the controller with different workload patterns. The simulated traffic arrives from the internet and is distributed by the load balancer on *Dom0* to the running Web servers. Having a 1 Gbps network link between the PMs avoided any network congestion while performing the experiments with high traffic load.

B. Main experiments

The following two main experiments were conducted:

- Resource conflict: Web service and batch, vertical scaling
- Resource conflict: Web service and batch, horizontal scaling

In the first scenario, a resource conflict between the Web service and batch-service occurs, and this leads to vertical scaling. The *spiky*-based workload pattern was used, starting from 0 and peaking at 1800 clients during a time-period of 5 minutes. The traffic load was high enough to make the Web server "steal" resources from the batch-service for a short amount of time. The autonomic controller measured the QoS of the applications in intervals of 5 seconds and 5 minutes, for the Web service and batch-service respectively.

The second experiment where a resource conflict led to horizontal scaling, was performed with the same settings as in the previous experiment. In addition, horizontal scaling is performed when the maximum number of resources is reached on the PM. In addition horizontal down-scaling is performed in relation to the traffic load. The workload is *trend*-based with increasing number of concurrent clients from 0 to 2700 during a time-period of 10 minutes.

C. Results

This subsection covers the results of the experiments.

1) Resource conflict: Web service and batch, vertical scaling: The results from the experiments with a resource conflict between the Web service and batch-service on a single PM is illustrated in Figure 7 and 8. The workload pattern for the experiment was *spiky-based* from 0 to 1800 clients during 5 minutes, which induces several fluctuation in traffic patterns. In addition a control interval of 5 seconds is used for the Web service.

There are two main traffic spikes in the response time, the first one reaching a peak of 900 ms, and the second spike rising up to 700 ms. The controller allocates resources within a short time after the traffic spikes, and manages to decrease the response time to the desired interval. The number of vCPUs is allocated based on the increasing response time and reaches a top of seven vCPUs. Before allocating seven vCPUs, the batch VM gets its number of vCPUs reduced by one in favor of the Web server so that it manages to keep the response time low. After 5 minutes the workload is finished and the Web server releases the vCPUs, which then are allocated to the batch VM to make up for the delayed execution of the encoding.

For the batch job, the FPS starts above 40 FPS and drops slowly to the desired interval. At the beginning there are two vCPUs allocated and this drops to one which is, as explained above, allocated in favor of the Web service VM. In the tenth minute, the control loop for the batch VM is run again and the FPS is right at the minimum desired FPS and a new vCPU is allocated. The FPS slowly increases and manages to recover from the delay in FPS between the fifth and the tenth minute.

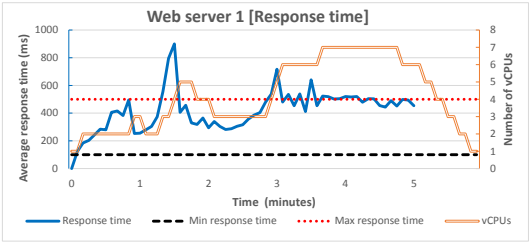

Fig. 7: Web server: response time in relation to vCPUs

The batch job finishes within the desired time-deadline of 25 minutes.

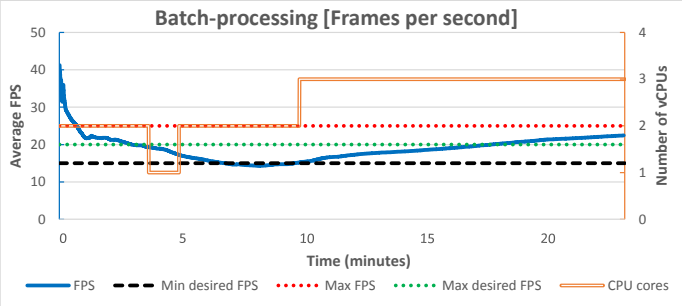

Fig. 8: Batch-processing: FPS in relation to vCPUs

2) Resource conflict: Web service and batch, horizontal scaling: In the second main experiment, a resource conflict between the Web service and the batch-service takes place leading to horizontal scaling. The *trend-based* workload pattern is simulated in this experiment from 0 to 2700 clients during 10 minutes, which means that the traffic increases up to 2700 simultaneous clients by the end of the test.

As illustrated in Figure 9 and 10, the response time jumps to 3000 ms and 4000 ms after three minutes, the number of vCPUs increases as a consequence of the spikes in the response time. Before allocating seven vCPUs, one core is removed from the batch VM, but still the response time is high and a new Web server is created on the second PM. The vCPUs then gets reduced to the half, and the second Web server is given the same number of vCPUs. Then the number of vCPUs increases on both of the Web servers and they manage to decrease the response time to the desired interval.

The batch VM starts right below 45 FPS, but the number of FPS is reduced as the Web server takes one vCPU. The core is allocated back at the next control interval for the batch job. However the FPS is below the desired value in the tenth minute and then another vCPU is allocated. The FPS increases steadily and manages to take back the lost calculated FPS. The batch job finishes within the desired time-frame of 25 minutes.

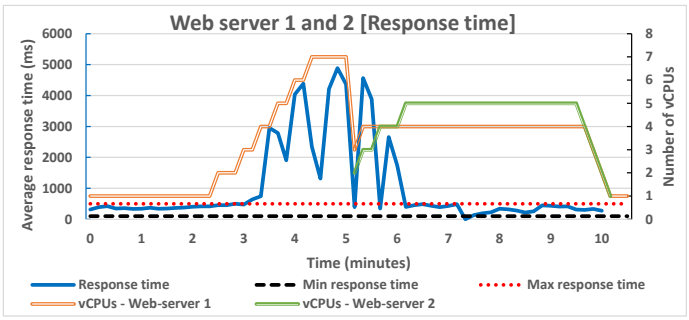

Fig. 9: Web server 1 and 2: response time in relation to vCPUs

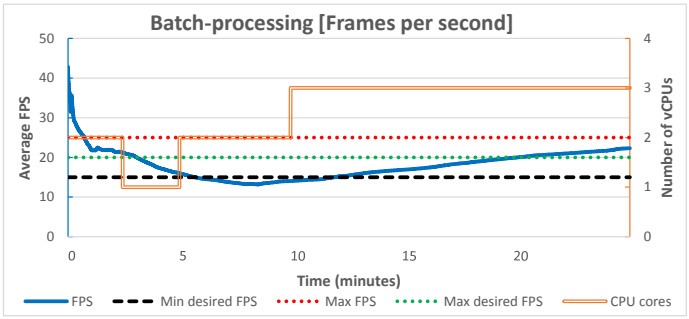

Fig. 10: Batch-processing: FPS in relation to vCPUs

D. Analysis

This subsection covers the analysis of the results of the two main experiments.

1) Resource conflict: Web service and batch, vertical scaling: In the first experiment with vertical scaling, the average response time for the Web service was 426 ms and 18.96 FPS for the batch job during the experiment, as shown in table V. The amount of requests which received HTTP response code 200 was 627 911, there were no requests that received the HTTP response code 400 or 500. 136 of the requests received timeout. Furthermore the amount data sent in the requests was 72.28 MB and received in responses was 1.52 GB.

TABLE V: Web server metrics with vertical scaling

Metrics	Web server
Average response time	426 ms
Average FPS	18.96 FPS
Response code: 200	627 911
Response code: 400/500	0
Timeout	136
Bandwidth - Sent	76.28 MB
Bandwidth - Received	1.52 GB

When it comes to the resource utilization, as illustrated in 11, the Web service had an 60% average utilization of memory, while the batch had an average memory utilization of 75.61% during the experiment. The Web service had an average CPU utilization of 86.32% during the experiments, while the batch on the other hand had an average CPU utilization of 89.34%.

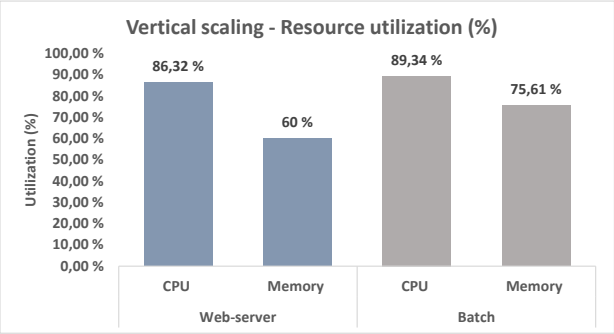

Fig. 11: Vertical scaling experiment: Average utilization of resources

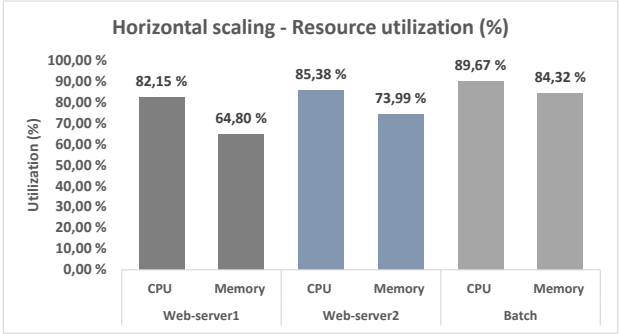

Fig. 12: Horizontal scaling experiment: Average utilization of resources

The average duration of violation of the target response time was 78.23 ms. The overall amount of responses which violated the SLA was 27.87%, while 71.67% of the requests were within the desired interval.

2) Resource conflict: Web service and batch, horizontal scaling: In the horizontal scaling experiment, as shown in table VI, the average response time during the experiment is 494 ms. The average FPS for the batch job is 18.65 and within the desired interval of 15- and 20 FPS. There were in total 789 992 requests which received HTTP response code 200, while there were no requests which revived HTTP response code 400/5000. Of the total amount of requests, 13 023 of them received timeout, *Loader* sent 97.83 MB data in requests and received 1.92 GB data in responses.

TABLE VI: Web server metrics with horizontal scaling

Metrics	Web server
Average response time	494 ms
Average FPS	18.65 FPS
Response code: 200	789 992
Response code: 400/500	0
Timeout	13 023
Bandwidth - Sent	97.83 MB
Bandwidth - Received	1.92 GB

The average amount of violations was 2463.94 ms in response time above the baseline. The amount of requests which violated the SLA-requirements was 26.23%, and 72.13% of the requests was within the desired interval. The utilization of resources is illustrated in Figure 12. The average utilization of CPU is higher on Web server 2 than Web server 1, the difference is 3.38%. The Web server 2 had a higher memory utilization of 73.99%, while the Web server 1 had an average memory utilization of 64.80%. The batch VM had higher utilization of both memory and CPU, 84.43% and 89.67%, respectively.

V. Conclusion

Due to the different nature of interactive and batch applications, coordinating their resource allocation is a challenging task. This paper tackles the problem of coordinating resource allocation for Web and batch services running under the same infrastructure. The controller used for the resource allocation is hybrid in the sense that it uses both resource usage and application level metrics for elasticity decisions. Moreover, while most resource allocation approaches solely concentrate on either vertical scaling or horizontal scaling, the proposed hybrid controller resorts to both vertical and horizontal scaling, simultaneously.

For the evaluation, a set of experiments were conducted, involving both vertical and horizontal scaling to achieve the desired QoS of the Web and batch applications. The experiments were run on two physical machines running Xen Hypervisor with the support for *hot-plugging* to tackle load bursts of heterogeneous applications. The response time of the Web service and the job progress of the batch service were used in decision making in order to efficiently provision resources to the applications. The results reveals that the proposed hybrid controller is able to achieve the desired target QoS for the Web service in all the experiments. Furthermore, the batch job managed to finish the workload within the desired deadline. The average utilization of CPU across all experiments is 83.70% for the Web service and 89.51% for the batch service, while the average memory utilization is 67.52% and 74.78%, respectively. Based on the experimental results, a combination of vertical and horizontal scaling seems effective in handling different types of load variations.

As a future work, we envision to resort to the theory of reinforcement learning for coordinating simultaneously the CPU and memory allocation in a more efficient manner according to the characteristics of the workload.

References

[1] W. Josh and D. Pierre. (2014, aug) Data center efficiency assessment. [Online]. Available: http://www.nrdc.org/energy/files/data-center-efficiency-assessment-IP.pdf

[2] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards energy proportionality for large-scale latency-critical workloads," in *Proceeding of the 41st Annual International Symposium on Computer Architecuture*, ser. ISCA '14. Piscataway, NJ, USA: IEEE Press, 2014, pp. 301–312.

[3] H. Ghanbari, B. Simmons, M. Litoiu, and G. Iszlai, "Exploring alternative approaches to implement an elasticity policy," in *2011 IEEE International Conference on Cloud Computing (CLOUD)*. IEEE, 2011, pp. 716–723.

[4] E. Lakew, C. Klein, F. Hernandez-Rodriguez, and E. Elmroth, "Towards faster response time models for vertical elasticity," in *2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing (UCC)*, Dec 2014, pp. 560–565.

[5] G. Moltó, M. Caballer, E. Romero, and C. de Alfonso, "Elastic memory management of virtualized infrastructures for applications with dynamic memory requirements," *Procedia Computer Science*, vol. 18, pp. 159–168, 2013.

[6] S. Farokhi, P. Jamshidi, E. B. Lakew, I. Brandic, and E. Elmroth, "A hybrid cloud controller for vertical memory elasticity: A control-theoretic approach," *Future Generation Computer Systems*, vol. 65, pp. 57–72, 2016.

[7] S. Farokhi, P. Jamshidi, D. Lucanin, and I. Brandic, "Performance-based vertical memory elasticity," in *2015 IEEE International Conference on Autonomic Computing (ICAC)*, July 2015, pp. 151–152.

[8] R. Buyya, C. S. Yeo, S. Venugopal, J. Broberg, and I. Brandic, "Cloud computing and emerging it platforms: Vision, hype, and reality for delivering computing as the 5th utility," *Future Generation computer systems*, vol. 25, no. 6, pp. 599–616, 2009.

[9] S. Farokhi, E. Lakew, C. Klein, I. Brandic, and E. Elmroth, "Coordinating cpu and memory elasticity controllers to meet service response time constraints," in *2015 International Conference on Cloud and Autonomic Computing (ICCAC)*, Sept 2015, pp. 69–80.

[10] S. Farokhi, "Quality of service control mechanism in cloud computing environments," Ph.D. dissertation, Vienna University of Technology, dec 2015.

[11] D. Lo, L. Cheng, R. Govindaraju, P. Ranganathan, and C. Kozyrakis, "Heracles: improving resource efficiency at scale," in *ACM SIGARCH Computer Architecture News*, vol. 43, no. 3. ACM, 2015, pp. 450–462.

[12] S. Spinner, N. Herbst, S. Kounev, X. Zhu, L. Lu, M. Uysal, and R. Griffith, "Proactive memory scaling of virtualized applications," in *2015 IEEE 8th International Conference on Cloud Computing (CLOUD)*, June 2015, pp. 277–284.

[13] L. Yazdanov and C. Fetzer, "Vertical scaling for prioritized vms provisioning," in *2012 Second International Conference on Cloud and Green Computing (CGC)*, Nov 2012, pp. 118–125.

[14] C. James. (2014, may) Forget application response time "standards" – it's all about the human reaction. [Online]. Available: http://me.riverbed.com/blogs/human-reaction-drives-application-response-time-standards.html