7-1-2013

Online Load Analysis For Automated Request-Quota Controlling In Clouds

Johannes Bendler
University of Freiburg, Freiburg, Germany, johannes.bendler@is.uni-freiburg.de

Markus Hedwig
University of Freiburg, Freiburg, Germany, markus.hedwig@is.uni-freiburg.de

Dirk Neumann
University of Freiburg, Freiburg, Germany, dirk.neumann@is.uni-freiburg.de

ONLINE LOAD ANALYSIS FOR AUTOMATED REQUEST-QUOTA CONTROLLING IN CLOUDS

Bendler, Johannes, Albert-Ludwigs-University of Freiburg, 79098 Freiburg, Germany, johannes.bendler@is.uni-freiburg.de

Hedwig, Markus, Albert-Ludwigs-University of Freiburg, 79098 Freiburg, Germany, markus.hedwig@is.uni-freiburg.de

Neumann, Dirk, Albert-Ludwigs-University of Freiburg, 79098 Freiburg, Germany, dirk.neumann@is.uni-freiburg.de

Abstract

The importance of efficient use of IT infrastructure capacities increases, as the growing amount of deployed IT systems increasingly requires resources. On the one hand, companies can outsource their IT demands to Cloud providers in order to only pay for the very amount of required computational power. On the other hand, cloud providers themselves need to be efficient as well, as it may be the deciding factor on their market positioning. This paper proposes a method to accept or refuse incoming requests based on a load estimation and load forecast. We will present the architecture and a general implementation of the so-called Quota-Control-Unit (QCU), followed by results of an extensive simulation. The evaluation will show that the introduced method is a promising approach of securing IT systems from overload while keeping the economic outcome at the possible maximum.

Keywords: Decision Support, Load Analysis, Cloud Computing.

Introduction

Nowadays, outsourcing of computational demands to Cloud systems or service networks increases. As companies' ecological awareness grows, they request highly flexible IT infrastructure to avoid any spare capacities that would be present only keeping in-house IT systems. Besides the ecological base, another factor that brings Cloud computing forward is the fact that only a minimum of IT infrastructure management effort is required when having all computational demands relayed to Cloud providers. Less spare in-house IT infrastructure leads to less unnecessary power-consumption and maintenance efforts. On the other side, companies that offer Cloud services (Infrastructure as a Service, Platform as a Service, and Software as a Service) for rent, based on the *"pay-as-you-go"* principle, need to be efficient as well.
Basically, Cloud computing is the aggregation of large amounts of hardware and delivering seemingly endless computational power to customers by massive use of virtualization. However, there are limits on provider side, since available hardware in fact is not endless. Providers have to properly scale their IT systems in order to satisfy the arbitrary demands of their customers, which is usually achieved by over-provisioning their hardware: large amounts of resources are running spare to cover eventual variation in customers' demands (Armbrust, et al., 2009). Since being efficient may be the deciding factor on providers' market positioning, the provisioning of resources on provider side has to be balanced between the satisfaction of customer demands and the amount of capacities running spare. In this paper, we deliver a system component, namely Quota-Control Unit (QCU), which aims to generate maximum possible revenue in case of customers' requirements exceeding the provider's capacities.

In case of a provider receiving urgent or short-term requests, respective to their Service-Level Agreements, it may be crucial to have them executed and answered as soon as possible. Should a provider accept more requests, if systems already run near overload? Not only by a high amount of incoming requests, but also by partial breakdown of infrastructure is the provider possibly faced with an overload situation. For example, in the morning of April 21st 2011, many Amazon cloud-based service providers were stricken to find their cloud services unavailable (AWS Tech Team, 2011). Should a provider faced with an overload situation still try to run all requests on narrowed resources? There is no universally valid answer, however, we can derive and deliver approaches to keep the generated revenue in these situations as high as possible. On provider-side, we assume requests to generate revenue when accepted and successfully executed or penalties when accepted but aborted or not answered in time. Based on this assumption, we can calculate the economic outcome for a provider by their system load and request acceptance in phases of heavy load. Tracking the load and its tendency can be important for early decisions on accepting or refusing incoming requests for the service provider, since the option of scaling up IT infrastructure is not available in case of partial breakdown or unexpected load peaks.

Thus, we tackle the outlined problem by searching for an approach of making informed decisions about whether new requests should be accepted by the provider.

This paper is structured as follows. First of all, the Related Work section takes existent approaches into account and analyzes proposed solutions. In the subsequent section, the Quota Control Unit (QCU) Model is introduced in detail. Afterwards, the Evaluation section describes the approach of an experimental study as a discrete event simulation using jobs and their requirements based on real workload traces. The simulation is executed under various parameter settings and its results are evaluated. The last chapter concludes the contents and findings and delivers open aspects that still should be reviewed for further work.

1 Related Work

This work combines various research threads that are relevant to the admission and management of job requests. This section points out aspects relevant to modeling a scenario, such as performance

modeling and workload forecasting, and addresses the technical challenges for systems like SOA (Service Oriented Architecture) and Clouds. Those models typically employ no or only naïve economic models that are feasible and result in monetary outcome.

Performance modeling is an active research field and has produced a variety of models as shown in (Urgaonkar, et al., 2008), (Cohen, et al., 2004). Furthermore, a detailed overview of how workload can be modeled is presented by (Feitelson, 2002), (Feitelson, 2011). An approach towards realization of high service levels and end-to-end Quality of Service (QoS) is the GLOBUS Architecture for Reservation and Allocation (Foster, et al., 1997), which uses advanced reservations to guarantee QoS. Specifically addressing the question of how job allocations affect resources, (Kournev, et al., 2007) introduce a framework for designing resource managers, using GLOBUS as a case study, that are able to predict the impact of a certain job on the overall system performance and to adapt the resource allocation in a way towards Service-Level Agreement (SLA) satisfaction. A model to optimize the system allocation for immediate use is developed by (Ardagna, et al., 2007). A good overview of SLAs that are commonly used in the field of Cloud Computing is provided by (Buyya, et al., 2009). Most of the newer contributions in this field extend the performance models to dynamic research management systems. For instance, the authors in (Gmach, et al., 2008) developed a reactive migration controller for virtualized environments. However, compared to our concept, their approach is only designed for basic single-layered systems. The paper by (Chandra, et al., 2003) introduces a resource allocation model for shared datacenters based on a queuing network performance model and a time series workload forecast mechanism. However, they do not consider SLAs in the provisioning process. Another concept Padala, et al., 2009) is an automated control model for virtual resources. The model manages the varying resource demands by dynamically allocating resources to or migrating virtual machines. Nevertheless, in modern cloud environments this migration approach is usually not supported. In (Ardagna, et al., 2009) a model has been developed to manage the resource demand of multiple concurrent systems. In contrast to our model, it optimizes the system only for a single point in time, rather than for the near future. The authors in (Lim, et al., 2010) developed an autonomic control model to scale elastic storage systems based on the utilization of the system.

Apart from technically oriented research streams, concerning revenue management, (Nair, et al., 2001) provides research results about the use of revenue management concepts by Internet Service Providers (ISPs). They consider the decision to accept or reject customers but did not take different service types or advanced reservation into account. Towards revenue generation, (Yeo, et al., 2007) show an approach for a pricing function depending on a base pricing rate and a utilization pricing rate.

The related work covers different aspects of related research problems. However, the proposed mechanisms mostly focus on resource allocation and do not take an upstream instance (e.g. placed in front of the request queue) into account that is able to filter requests before they enter the certain system.

2 Quota Control Unit Model

As previously mentioned, the problem we tackle in this research is the lack of an upstream instance able to filter incoming requests before they enter the certain system. We address this problem by using the well-known design science research approach (Hevner, et al., 2004) to develop an IT artifact that allows us to measure and compare upstream decision performance in a confidential manner. The design science methodology seeks to create IT artifacts that are intended to solve specific organizational problems and provide rigorous evaluation of these artifacts based on utility rather than an empirical test of theories. This encompasses successive steps of problem identification, definition of objectives for a solution, design or development of a suitable IT artifact, and demonstration of the proof of concept, evaluation and communication (Hevner, et al., 2004).

In order to gain the ability of analyzing and controlling incoming requests, as it is the intention of the QCU, it has to be properly positioned within an IT system. Most IT systems that may receive demand spikes provide a central request interface (as the web frontend for an online-shop, for example), because they are laid out for application of load balancers and dynamically scaled infrastructure. To gain

full control over incoming requests, the QCU is positioned as an upstream instance even before requests reach the central interface in our setup. It may be reasonable to place the QCU behind an available load-balancer though, when it comes to multi-tier architectures or specialized hardware for certain tasks. Figure 1 outlines the QCU positioning.

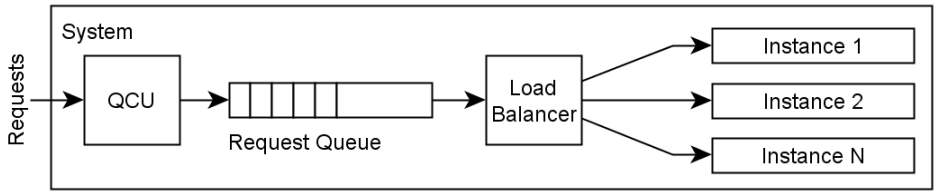

Figure 1. Proposed QCU Positioning

Based on the assumptions outlined above, several basic QCU requirements can be derived. First of all, the QCU has to be aware of the overall system load, e.g. the information about the current workload being above maximum capacity or not is required.

Requirement 1: The QCU needs to include an estimation of the current system load.

A decision only based on the very current system load may be misleading, thus the QCU should be able to calculate the trend of the system load, such that it could already start accepting more jobs if the system load is slightly decreasing though still near to full capacity, for example.

Requirement 2: The QCU needs the capability to identify the system load's tendency.

Since the QCU should be applied to make a decision about the amounts of requests that shall be handled in order to strictly avoid any overload situation, it should contain some metric for acceptance ratio finding.

Requirement 3: The QCU has to support an informed choice of job refusals.

Towards fulfillment of the requirements, the QCU model proposed in this work consists of mainly four aspects (cf. Fig. 2). Essentially, the QCU:

i. estimates the average load in the considered time slot,

ii. calculates a weighted and smoothed tendency value

iii. derives the tendency function to calculate a trend,

iv. and applies a quota-based metric as decision basis for defining the part of the new jobs that shall be accepted in the current time slot.

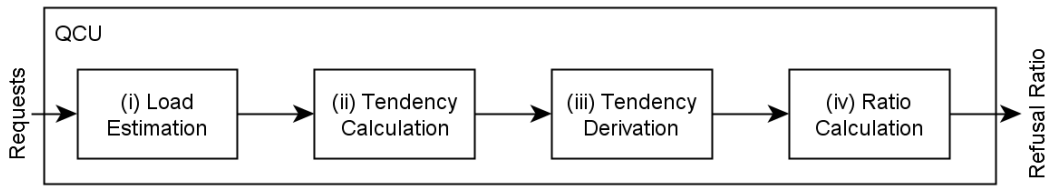

Figure 2. QCU Process Pipeline

According to the introduction above, the QCU's four main aspects are described in the following four sub-sections. Table 1 lists required variables and parameters. Please note that the QCU properties introduced here shall only serve as an orientation towards an efficiently working QCU. The simulation framework applied later on for evaluation allows full modification or replacement of any of the modules as well as free parameter modification. This paper considers the two resource types CPU and Memory exemplarily. However, the set of available resources R can be arbitrarily extended based on the requirements of the application scenario.

Variable	Domain	Description
h	$h \in \mathbb{N}, h > 2$	Horizon; Amount of time slots the QCU is looking into the past to calculate the tendency.
t	$t \in \mathbb{N}$	Time slot; A certain position within system runtime.
S_t		Set of Services; Contains all Services from within the system that are currently pending or running, dependent on time slot.
R	$\{\text{CPU}, \text{Memory}, \dots\}$	Resource; Set of available resources. This work only uses `CPU` and `Memory`.
$\text{level}_\text{t}(r)$	$r \in R$ $\text{level}(r) \rightarrow [0,1]$	Capacity utilization; Level of usage of the certain resource between 0 (no usage) and 1 (used to capacity).
λ_t	$\mathbb{R}$	Load; Estimated characteristic of the system load, dependent on time slot.
τ_t	$\mathbb{R}$	Tendency; Calculated, smoothed tendency in time slot t of the system load over the past h time slots.
ρ_t	$[0,1]$	Refusal ratio; Quota of jobs in time slot t that are refused and hence not added to the system.

Table 1. QCU Variable Definition

The upcoming sections make use of some key terms that require introduction. First of all, the term **load** refers to the overall system utilization, i.e. it is larger than or equal to zero. since in this approach, it is calculated by scaling the average level of resource utilization by the amount of service requests in the system, it may become larger than 1. The exact formula is delivered in section 2.1, equation (1). The next sections use formulae and terms that refer to **time slots**, denoted t. Generally speaking, a time slot can represent any time span, only dependent on the desired resolution of calculations. These calculations are performed in discrete steps, related to each other in a timed order. For simulation purpose, time slots do not have a fixed width - in case of calculations becoming more complex, they are completed before the simulation proceeds to the next time slot, such that a real-time system can be modeled and simulated (time-criticalness is eliminated).

2.1 Load Estimation

As the QCU will calculate a load tendency later on, the overall load estimation is required at any time. Depending on the certain setup, productive environments may deliver direct system load observation at runtime. If this is the case, the live load monitoring can be utilized instead of the estimation step presented in this section. In this work, we assume the system to not deliver live load monitoring. Furthermore, all resources are presented abstractly in a single resource pool where services may satisfy their requirements when launched. As listed in table 1, we do only focus on the two resources CPU and Memory. Nonetheless, this approach will work with various resource pools and a complex distribution of services to them, but for the basic goal the narrowed approach we apply is satisfactory. The set of all available resources R can be arbitrarily extended if desired, since equation 1 normalizes the average of all resources by the amount of resource types considered. Our QCU calculates the current average system load by multiplying the amount of available (e.g. running or pending) services by the mean of all resource usage levels, respective to equation 1. Currently, the estimated load is scaled by the total amount of services. Future research will distinguish between pending and running services since scheduling services may lead to

$$\lambda_t = \frac{\sum_{r \in R} \text{level}_\text{t}(r)}{|R|} \cdot |S_t| \tag{1}$$

In this approach, we explicitly decided to use the resource load average for load estimation instead of the maximum over all resources due to the following considerations. Though equation (1) may result in an underestimation of resource usage, because the arithmetic mean over all resource levels is calculated, it is satisfactory for the load extrapolation following below. As another approach, the maximum resource usage level over all types of resources could be selected to estimate the current system load, but since resources may have different impacts on the computational capacities of the entire system,

this may lead to overestimation of resource usage. In case of load overestimation, the cloud system could only hardly be utilized up to full capacity and would always have spare computational power that remains unused. However, this difference between the two possibilities of load estimation will be regarded closer in future research.

2.2 Tendency Calculation

Basically, the tendency is defined as the difference between estimated loads in sequential time slots. This alone is not satisfactory, since the estimated load may be quite noisy and thus the calculated tendency may eventually become inaccurate. In order to fix this issue, tendency values are calculated by including previous results in a time series approach with moving average, weighted by their time distance. Additionally, they are smoothened afterwards.

$$\tau_t = \frac{1}{h} \cdot \sum_{i=1}^{h} \frac{\lambda_t - \lambda_{t-i}}{i} \tag{2}$$

According to equation 2, the last load values within the horizon are used to calculate the mean tendency of the load in time slot t. The factor $\frac{1}{h}$ scales the result by the horizon size. Note that equation 2 uses differences between old load values and the current estimation to calculate the mean slope and hence results in a tendency. For further smoothing, the resulting tendency value and all its predecessors within the given horizon are averaged. The result is the tendency value in the current time slot.

2.3 Tendency Derivation

Now that the averaged tendency is available, its slope can be calculated by derivation. Using the derivation to estimate the system load trend is a proper way for automated decision support, since it can be directly compared to zero in order to check whether tendency is going up or down and at which slope steepness. In order to approximate the derivation in a certain point based on discrete values, the function usually needs to be estimated first. As we do not know which polynomial degree may fit best because load values can be arbitrarily various concerning their flow, only a linear regression over the past tendency values is applicable.

$$\tau'_t = \lim_{\epsilon \to 0} \frac{\tau_{t+\epsilon} - \tau_t}{\epsilon} \tag{3}$$

Equation 3 delivers the approximation of the first derivative in a certain time slot. Basically, we perform a linear regression based on the horizon and linear extrapolation prior to the right-hand side approximation (eq. 3) towards the derivation. This extrapolation sorts out the problem of time slots being discrete values and ϵ being continuous.

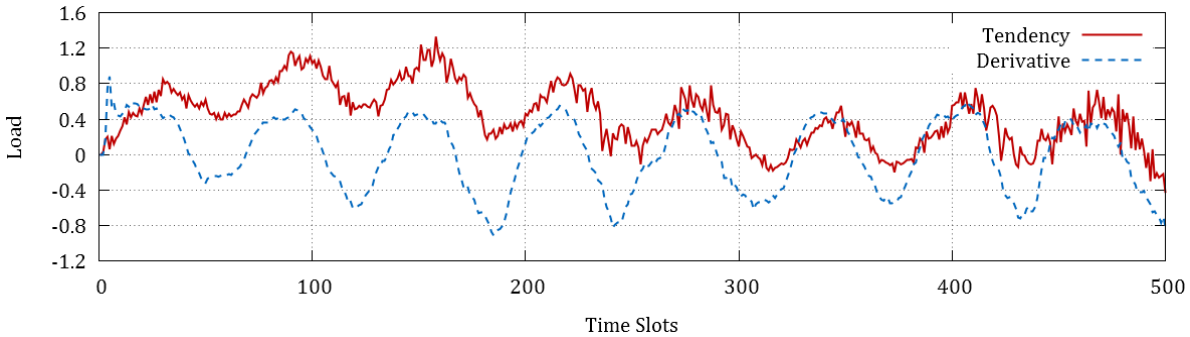

Figure 3: Example of Tendency and its Derivative without Job Refusals

Figure 3 shows an example of a tendency trace in a frequently overloaded setup and its approximated derivative without actually making a decision (e.g. all incoming jobs are accepted). The upper red (solid) graph is a calculated tendency trace based on results from a discrete event simulation. The

lower blue (dashed) graph is its derivative representing slope steepness (the offset in Figure 3 emerges from the use of a horizon h and corresponds to its size in time slots). Comparing the derivative to zero, one can clearly identify the trend of the current system load.

2.4 Ratio Calculation

The goal of the decision within the QCU is to flatten the load value graph, e.g. to have it constantly near the maximum possible load, but strictly avoid phases of further system overload. Hence, the most important part to focus on is the derivative being positive. Since the QCU does not know the maximum system capacity that is available, the decision must not only be based on the derivative being positive or not. We have to include values from the past in order to allow increasing system load up to a certain level. A good decision would result in the derivate graph oscillating around zero and thus keeping the system at a stable load level.

$$\rho_t = \begin{cases} \dfrac{\tau'_t}{\max\limits_{x \in h} \tau'_x} & \text{for } \tau'_t > 0 \\ 0 & \text{for } \tau'_t \leq 0 \end{cases} \tag{4}$$

According to equation 4, the current tendency value is compared to the maximum tendency value in the horizon, if the derivative is larger than zero. If the current tendency value is the maximum from all tendencies within the horizon, all new jobs in the current time slot are refused. Only if the system relaxes, more and more jobs are accepted. If the tendency is below zero, all jobs are accepted. This leads to an oscillation around zero as shown in Figure 4. Please note, that the QCU is not responsible for assigning priorities to each incoming request. In case of priorities being present, the QCU can easily be extended to first sort incoming requests by their respective priorities and then to refuse them starting at the request with lowest priority. For this work, we treat every request equally.

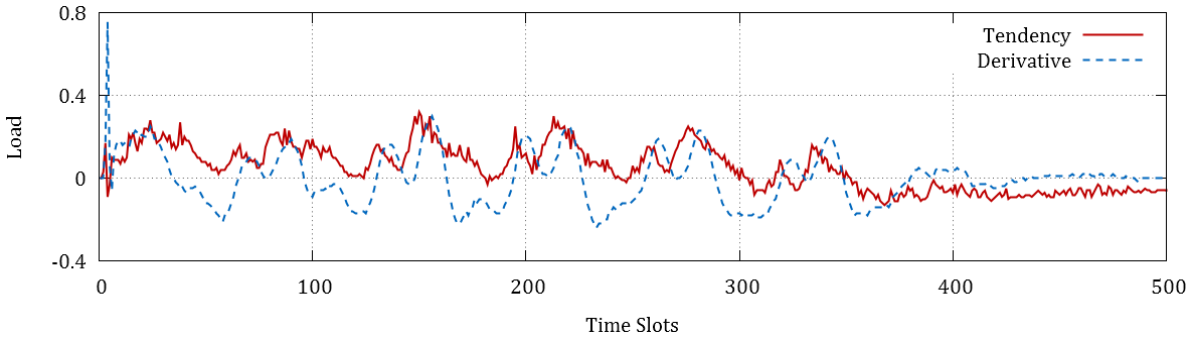

Figure 4. Example of Tendency and its Derivative with Job Refusal Applied

3 Evaluation

In order to test and evaluate the QCU model described above, a discrete event simulation is chosen. The problem's parameters are too manifold to perform a complete calculation towards the mathematical problem. Hence, we cannot provide a mathematically proven verification but only derive tendency results. The basic simulation flow is outlined in Figure 5. A generator constantly feeds the QCU with new jobs, which then decides - based on resource usage and amount of running services - which part of the new jobs to accept and which to refuse. Refused jobs are immediately dropped from the system; accepted ones are added to the set of currently pending jobs. The scheduler prioritizes the pending jobs by a certain policy (for example their expected revenue or simple first-come first-serve) and starts as many of their services as possible, restricted by the amount of available resources in a central resource pool. Jobs that reside in pending state for too long are aborted as they are assumed to violate Service-Level Agreements. Services in running state lock their allocated resources for a certain amount of steps, then release them and drop from the system as finished services.

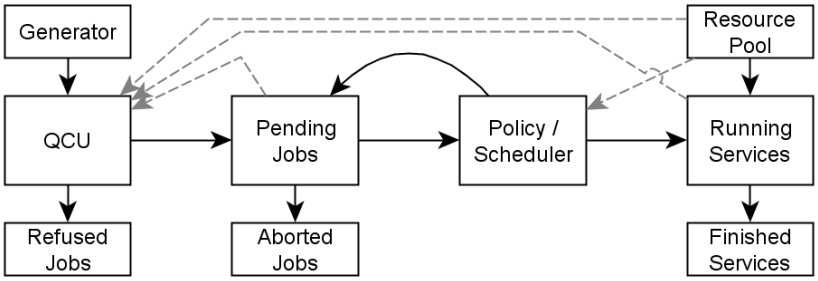

Figure 5. Simulation Flow Overview

3.1 Data Sources

In our simulation, jobs are assumed to be a sequence of services with the option of concurrent execution of grouped services (as service tuples). As an example, a job consisting of three services where the first two services may be executed concurrently but the last one may only start on successful termination of its predecessors, it would be notated as `[(A, B), (C)]`. This assumption shifts the simulation towards a more general applicability since jobs are not atomic but defined as a sequence of atomic services. Furthermore, each service requires a certain amount of available resources to be launched and requires a certain amount of time slots to be finished.

The properties delivered in Table 2 aim to create a general scenario and thus are tightly geared to the findings from the A*STAR data traces (e.g. those job signatures from the traces that deliver CPU and Memory requirements). While a Gaussian distribution is assumed for capacity requirements (CPU and Memory), the composition of services to jobs is based upon uniform distribution over a selected range. Job templates serve as the basis for jobs that are spawned live by the generator. The valuation of a service is geared to the cost of an Amazon EC2 instance and since these are prices per CPU hour, the bid is also dependent on the service's CPU requirement. Though penalties are usually defined in Service Level Agreements (SLAs), they are drawn from a Gaussian distribution in this work in order to minimize the amount of critical assumptions. However, variety is generated by placing the penalty's μ-value on the certain request's bid value and allowing a σ of $\frac{1}{4}$ of the bid value.

`CPU` required per service	Gaussian, $\mu = 13.3, \sigma = 11.85$
`Memory` required per service	Gaussian, $\mu = 4{,}600, \sigma = 3{,}700$
Amount of job templates per scenario	Uniform in $[4, 12]$
Amount of service tuples per job template	Uniform in $[2, 4]$
Amount of services per service tuple	Uniform in $[1, 4]$
Valuation (bid) of a service	Gaussian, $\mu = 0.95, \sigma = 0.3$, multiplied by `CPU`
Penalty of a service	Gaussian, $\mu = \text{bid}, \sigma = \frac{\text{bid}}{4}$

Table 2. Simulation Parameter Distribution

All in all, the values are chosen in a manner to easily generate system overload, but as well have the option to have the system run below its maximum capacities. In our simulation environment, the above mentioned distribution-parameters are fully adjustable, and even hand-written scenarios are possible. This is required when it comes to an extensive simulation with many different parameter values.

3.2 Results

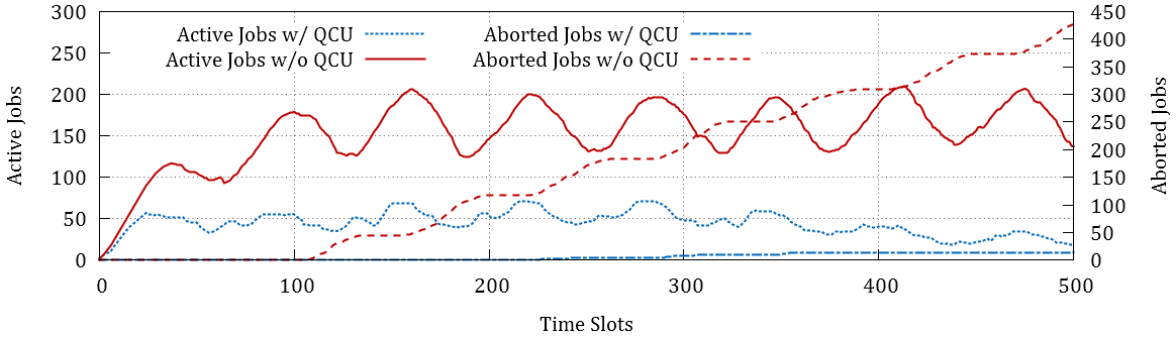

Figure 6. Load Comparison

Figure 6 compares the simulated system load without QCU (red solid graph as actual load, red dashed graph as accumulated amount of aborted jobs) to the simulated load when a QCU is applied (blue dotted graph as actual load, blue dash-dotted graph as accumulated amount of aborted jobs). Having a more detailed look to the graphs that represent the simulation without a QCU (red graphs, solid and dashed) one can recognize the system *constantly residing in an overloaded state*, because the amount of aborted jobs is constantly increasing. Steeper slopes can directly be linked to the load spikes several time slots prior to them. This means, the system without a QCU is unable to complete all incoming jobs and many of them are aborted as they are waiting too long and violate Service-Level Agreements.

The graphs representing the simulation flow where a QCU was applied (blue graphs, dotted and dash-dotted), the first thing to notice is that the average load of the system with QCU is significantly lower than the average load of the system without QCU. With this in mind and the fact that there are very few jobs aborted during simulation with a QCU applied, one can derive that the system is not being overloaded. Thus, the QCU cancels incoming job requests according to its current and maximum load quite well. However, since the QCU only estimates load value and derives a basic tendency, it does not refuse the perfect amount of jobs. Perfect amount of job refusal would match the amount of aborted jobs in the scenario without a QCU as this number represents the overhead of requests that led to system overload. The application of a QCU resulted in refusal or abort of overall 608 requests, while the system without a QCU only aborted 426 jobs. The QCU cancels about 42.72% more jobs than the system with no QCU applied had to abort due to overload. Table 3 delivers the values that are shown graphically in Figures 6 and 7 and adds extra measures.

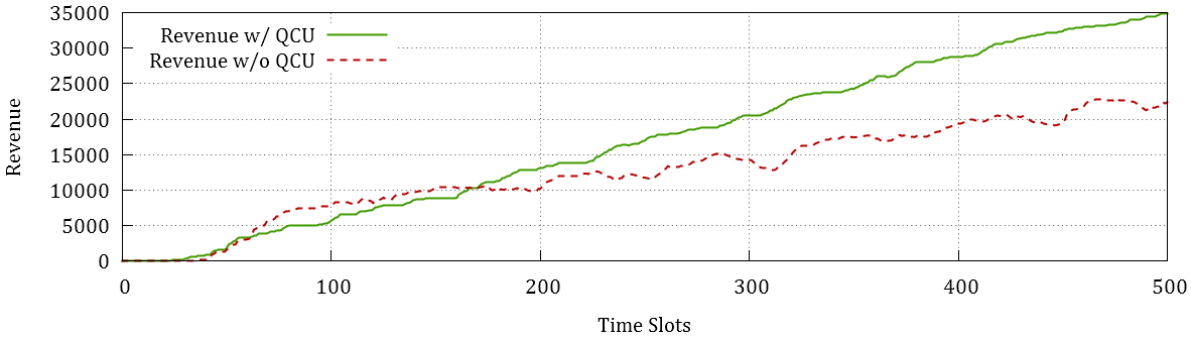

Figure 7. Revenue Comparison

In Figure 7, the accumulated overall generated revenue is outlined. The red dashed graph shows the revenue of the system with no QCU, the green solid graph shows the revenue of the system with a QCU applied. The revenue value is calculated by the difference between total bid value of all jobs that were accepted and the penalty values of all jobs that were accepted but aborted. From time slot 50 to time slot 170, the system without QCU performs slightly better, e.g. generates higher revenue. Starting at time slot 170 and ongoing, the system with QCU outperforms the non-QCU simulation. The slight

advantage of the non-QCU system (in the time slots mentioned above) results from the QCU having to level off after simulation has started. Furthermore, the system without QCU is not yet at overload in the concerned time slots and thus generates higher revenue as no jobs are canceled or aborted (cf. Figure 6, respective time slots: from time slot 100 on, jobs are being aborted by the non-QCU system). As soon as jobs are aborted in the non-QCU system, the according revenue graph's slope decreases. In contrast to that, the system with QCU applied has a constantly increasing revenue value. The final revenues drawn from our simulations are 22198 for the non-QCU system and 34904 for the system with QCU, which is an increase by 57.24%. Please note, that these values are measured in fictional units and to not reflect real money outcome but rather a comparison and trend of the systems with or without QCU depending on the generated load.

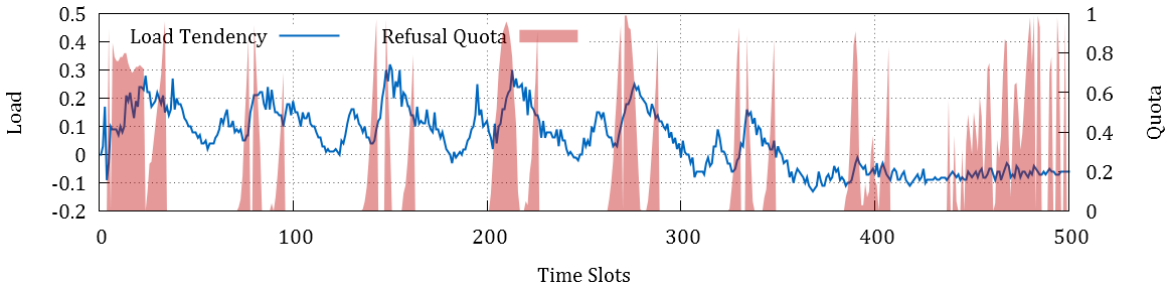

Figure 8. QCU Decision on Refusal Quota

The QCU decisions over the first 500 steps are outlined in Figure 8. The calculated load tendency, based on the overall system load estimation by the QCU, is represented by the blue graph that refers to the left y-scale. The QCU's request refusal quota is shown as a red filled curve scaled between 0 and 1 on the right y-scale. The figure shows that the refusal quota experiences peaks whenever load tendency spikes are detected. After the QCU has leveled off at around time slot 400, the system load tendency becomes stable with a slight upward trend. At this point, a heavy quota fluctuation can be observed, because the system load has leveled and the QCU's tendency derivation oscillates around zero to keep the system state stable. As a long-term view, Figures 9 shows how the system has leveled and operates at a constant load. The load tendency value (blue solid graph) fluctuates around zero.

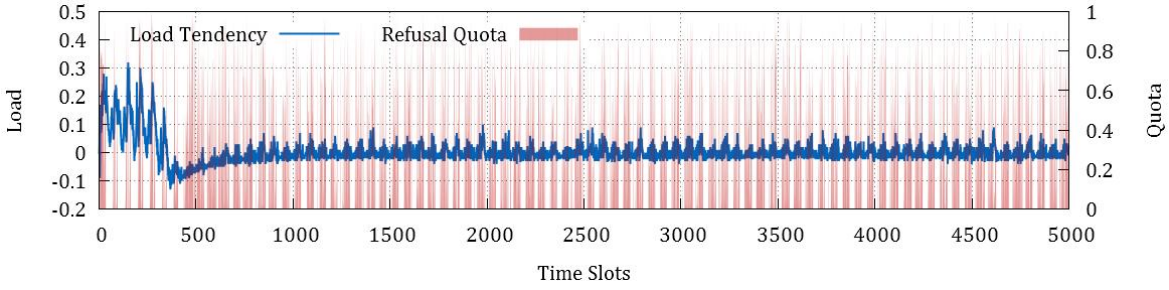

Figure 9. QCU in Long-Term View

Table 3 shows the accumulated results after 500 time slots in comparison to the results after 5000 time slots. Since the QCU's required level-off phase after initialization is quite heavy related to the first 500 time slots (cf. Figure 9), the refusal overhead (jobs being refused by QCU that would not have been aborted without QCU) is quite high. After the system has reached its level, the refusal overhead shrinks down to not more than 30% within 5000 time slots. The same applies for the revenue increase being only around 60% after 500 time slots but rising up to over 70% in 5000 time slots.

	500 Time Slots		**5000 Time Slots**	
Measure	No QCU	QCU	No QCU	QCU
Aborted jobs	426	14	5213	14
Refused jobs	0	594	0	6678
Refusal overhead by QCU to non-QCU		42.72%		28.37%

Overall revenue	22198	34904	185038	315968
Revenue increase by QCU to non-QCU		57.24%		70.76%

Table 3. Accumulated Results

The revenue increase strongly depends on the amount of new requests fed into the simulation in each time slot and hence is not to be read as an absolute economic improvement. However, it shows that the application of a QCU affects the monetary outcome positively in case of system overload.

4 Conclusion

Being faced with a single multi-tier architecture system that centrally accepts requests and delegates them to special hardware internally by application of load balancing requires to take an upstream instance (placed in front of the request queue) for request filtering into account, which is able to accept or refuse incoming requests based on the estimated current system load and its tendency. We proposed our QCU concept as an upstream filtering instance as described. We have shown that the appliance of a QCU leads to a bonus towards economic and monetary outcome of a system under heavy load, e.g. general overload due to numerous requests or unexpected overload by resource outage. Referring to the initially introduced requirements for the QCU instance, our simulations and evaluations have shown that the provided QCU concept is able to cover all of them. It tracks the system load and calculates its tendency by linear extrapolation over a moving average process. Furthermore, it is able to derive the slope of the tendency to use it as a forecast mechanism. Based on this forecast, the QCU can make informed decisions on the quota of requests that should be refused to avoid a system overload.

In order to measure the overall QCU performance, several simulations were performed. As a next step, an implementation is planned for extensive tests in a labor environment. Our simulations reveal the economic value of a QCU if applied, as they deliver the following findings:

1. The system with a QCU applied still runs near maximum capacity; the QCU does not decline more jobs than required. This means that the QCU is close to an optimal solution of the trade-off between request refusal and system load.
2. The amount of accepted requests that had to be aborted has significantly decreased in comparison to a system that did not use a QCU. Thus, the appliance of a QCU significantly reduces penalty payments due to violation of Service-Level Agreements or Quality of Service.
3. The overall revenue was lifted considerably in a long-term run when a QCU was used in comparison to a system without a QCU when requests are generated to provoke system overload.

Even though our straightforward model exhibits desirable results, there are possibilities to improve the QCU and to adapt it to meet additional system requirements. Apparently, the prediction is only based on the derivation of the weighted and smoothed tendency of an estimated system load. Using a more general procedure to state a system's load may greatly improve the results. As already mentioned in section 3.1, the effect of overestimation of resource usage levels should be carefully compared to the consequences of resource underestimation. This means reviewing the monetary tradeoff between parts of computational capacities remaining unused (due to overestimation) and potentially more rejected requests (due to underestimation). Furthermore, enforcement of the refusal rate does not take eventually contracted QoS (Quality of Service) levels for certain customers into account. A valuable extension of the QCU concept could be the introduction of request prioritization based on expected profit (profit heuristics) when applying the refusal ratio. Smoothing of the tendency graph currently is performed by averaging the neighborhood (moving average process). Applying a mathematical process of convolution should lead to a graph with far less noise. Furthermore, instead of using the first derivative, a more precise forecast mechanism could be applied, such as wavelets or Fourier-based forecasting.

References

Ardagna, D., Trubian, M. and Zhang, L., 2007. "SLA Based Resource Allocation Policies in Autonomic Environments", in *Journal of Parallel and Distributed Computing*, 67(3), pp. 259-270. Available at: http://dx.doi.org/10.1016/j.jpdc.2006.10.006

Ardagna, D. et al., 2009. "Run-Time Resource Management in SOA Virtualized Environments", in *Proceedings of the 1st International Workshop on Quality of Service-Oriented Software Systems - QUASOSS '09*, p. 39. Available at: http://portal.acm.org/citation.cfm?doid=1596473.1596484

Armbrust, M., et al., 2009, "Above the Clouds: A Berkeley View of Cloud Computing", UC Berkeley Reliable Adaptive Distributed Systems Laboratory

AWS Tech Team, „Summary of the Amazon EC2 and Amazon RDS Service Disruption in the US East Region," 2011. Available at: http://aws.amazon.com/message/65648

Buyya, R. et al., 2009. "Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility", in *Future Generation Computer Systems*, 25(6), pp.599-616. Available at: http://dx.doi.org/10.1016/j.future.2008.12.001 [Accessed April 30, 2011].

Chandra, A., Gon, W. and Shenoy, BP., 2003. "Dynamic Resource Allocation for Shared Data Centers", in *Quality of Service - IWQoS 2003*, vol. 270, pp. 381-398. Available at: http://www.springerlink.com/content/h56r570l4u707466

Cohen, I. et al., 2004. "Correlating Instrumentation Data to System States: A Building Block for Automated Diagnosis and Control", in *OSDI*.

Feitelson, D.G., 2002. "Workload Modeling for Performance Evaluation", in *Lecture Notes in Computer Science*, 2002, Volume 2459/2002. Available at: http://www.springerlink.com/content/xpx7gt5d7egwv3vm

Feitelson, D.G., 2011. "Workload Characterization and Modeling Book", Available at: http://www.cs.huji.ac.il/~feit/wlmod/.

Foster, I., Kesselman, C., 1997. "GLOBUS: A Metacomputing Infrastructure Toolkit", in *International Journal of High Performance Computing Applications*, June 1997, vol. 11, no. 2, pp. 115-128.

Gmach, D. et al., 2008. "Adaptive Quality of Service Management for Enterprise Services", in *ACM Trans. Web*, 2(1), pp. 1-46.

Hevner, A.R., March, S.T., Park, J., Ram, S., 2004. "Design Science in Information Systems Research", in MIS Quarterly, 28(1). Society for Information Management and The Management Information Systems Research Center Minneapolis, MN, USA.

Kounev, S., Nou, R., Torres, J., 2007. "Building Online Performance Models of Grid Middleware with Fine-Grained Load Balancing: A GLOBUS Toolkit Case Study", in *EPEW'07 Proceedings of the 4th European Performance Engineering Conference on Formal Methods and Stochastic Models for Performance Evaluation*. Berlin, Heidelberg: Springer-Verlag.

Lim, H.C., Babu, S. and Chase, J.S., 2010. "Automated Control for Elastic Storage", in *Proceeding of the 7th International Conference on Autonomic Computing*. New York, NY, USA: ACM, pp. 1-10. Available at: http://doi.acm.org/10.1145/1809049.1809051

Nair, S.K., Bapna, R., 2001. "An Application of Yield Management for Internet Service Providers", in *Naval Research Logistics*, vol 48, issue 5, pp. 348-362, August 2001.

Padala, P. et al., 2009. "Automated Control of Multiple Virtualized Resources", in *Proceedings of the fourth ACM European Conference on Computer Systems - EuroSys '09*, p. 13. Available at: http://portal.acm.org/citation.cfm?doid=1519065.1519068

Urgaonkar, B. et al., 2008. "Agile Dynamic Provisioning of Multi-Tier Internet Applications", in *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 3(1), Available at: http://portal.acm.org/citation.cfm?id=1342172.

Yeo, C.S., Buyya, R., 2007. "Pricing for Utility-Driven Resource Management and Allocation in Clusters", in *International Journal of High Performance Computing Applications*, vol. 21, no. 4, pp. 405-418