Model-Based Deadtime Compensation of Virtual Machine Startup Times

Dellkrantz, Manfred; Dürango, Jonas; Robertsson, Anders; Kihl, Maria

2015

Model-Based Deadtime Compensation of Virtual Machine Startup Times

Manfred Dellkrantz
Dept. of Automatic Control
Lund University
Sweden
manfred@control.lth.se

Jonas Dürango
Dept. of Automatic Control
Lund University
Sweden
jonas@control.lth.se

Anders Robertsson
Dept. of Automatic Control
Lund University
Sweden
andersro@control.lth.se

Maria Kihl
Dept. of Electrical and
Information Technology
Lund University
Sweden
maria.kihl@eit.lth.se

ABSTRACT

Scaling the amount of resources allocated to an application according to the actual load is a challenging problem in cloud computing. The emergence of autoscaling techniques allows for autonomous decisions to be taken when to acquire or release resources. The actuation of these decisions is however affected by time delays. Therefore, it becomes critical for the autoscaler to account for this phenomenon, in order to avoid over- or under-provisioning.

This paper presents a delay-compensator inspired by the Smith predictor. The compensator allows one to close a simple feedback loop around a cloud application with a large, time-varying delay, preserving the stability of the controlled system. It also makes it possible for the closed-loop system to converge to a steady-state, even in presence of resource quantization. The presented approach is compared to a threshold-based controller with a cooldown period, that is typically adopted in industrial applications.

1. INTRODUCTION

1.1 Background

Cloud computing has in the recent years become the standard for quickly deploying and scaling Internet applications and services, as it gives customers access to computational resources without the need for capital investments. In the Infrastructure as a Service (IaaS) service model, cloud providers rent resources to customers in the form of physical or virtual machines (VMs), which can then be configured by the customers to run their specific application. For a cloud customer aiming at providing a service available to the public, this poses the challenge of renting enough resources for the service to remain available and provide high quality of service (QoS), and the cost of allocating too much resources. Pair this with a workload that is time-varying due to trends, weekly and diurnal access patterns and the challenge becomes more complex.

For this reason, to cope with varying load, cloud services often make use of *autoscaling*, where decisions to adjust resource allocation are made autonomously based on measurements of relevant metrics. There is currently a plethora of different autoscaling solutions available, reaching from simple threshold-based to highly sophisticated based on for example control theory or machine learning. The solutions are commonly categorized as either reactive or proactive to their nature. In the former case, decisions are based on current metric measurements relevant to the load of the cloud service, while in the latter case on a prediction of where the metrics are heading.

Both approaches have in common that they usually do not distinguish between cases where the metrics are only indirectly related to the actual QoS of the cloud service, such as the arrival rate, or metrics that are directly coupled to the QoS, such as response times. From a control theoretical point of view, we could therefore further categorize the first case as feedforward approaches and the second case as feedback approaches. Feedforward control schemes can in many cases give good performance, but generally requires excellent apriori knowledge of the system to be controlled, and lack the ability to detect any changes or disturbances that affect the system. Feedback solutions on the other hand are generally more forgiving when it comes to system knowledge requirements. They can also compensate for unforeseen changes since they base their decisions on metrics directly related to the QoS.

For cloud services, decisions to add more resources usually requires starting up a new VM. This in turn means that the cloud provider needs to place the machine, transfer the OS data it needs and boot it up. Overall, the time from decision to a VM to get fully booted typically ranges from a few tens of seconds up to several minutes [12]. The long time

delays this leads to are an inherently destabilizing factor in feedback control. The key reason is the following: long time delays from a scale up decision to a full actuation prompts the feedback controller to continue commanding increased resource provisioning due to the fact that it cannot yet see the effect of its earlier decisions.

In practice, these time delays need to be considered when designing feedback based autoscaling solutions in order to avoid destabilizing the closed loop system. Possible existing solution include having a low gain in the feedback loop, essentially making the autoscaler very careful with continuing adding more resources before the effect of past decisions start showing up. Another solution is to implement a so-called *cooldown* period, as implemented in [1, 2, 3]. In autoscalers employing cooldown, any decision to scale resources activates the cooldown period, during which subsequent scaling attempts are ignored.

In the current paper, we take a different approach and adopt a solution that has similarities to the Smith predictor, a technique commonly used in control theory for controlling systems with long time delays. In essence, the Smith predictor works by running a model-based simulation of the controlled system without the delays, and use the outputs from this simulation for feedback control. Only if there is a deviation between the true system output and a delayed version of the simulated output are actual measurements from the real system used for control.

1.2 Related work

As cloud computing has grown more popular, the autoscaling challenge has attracted attention and resulted in numerous proposed solutions, for example [17, 9, 14]. A thorough review of existing autoscaling solutions can be found in [11]. The level at which reconfiguration delays are explicitly considered in existing autoscaling solutions varies depending on the underlying assumption of the magnitude of the delays and choice between feedforward and feedback control structures. Ali-Eldin *et al* [5] use an approach where scaling down is done reactively and scaling up proactively, but otherwise assumes that any reconfiguration decision is actuated immediately. Similarly, Lim *et al* [10] design a proportional thresholding controller with hysteresis where a feedback loop from response times to the number of allocated VMs is closed. Also here the assumption is that VMs can be started instantaneously.

Berekmeri *et al* [6] use an empirically identified linear time-invariant model with a time delay to design a controller for deploying resources in a MapReduce cluster to handle incoming work. The time delay corresponds to the reconfiguration delay and is assumed to be constant. As shown by Mao *et al* [12], VM startup times can vary heavily, both depending on application and infrastructure.

In Gandhi *et al* [8] the authors identify reconfiguration delays as the main reason for poor performance in many reactive and proactive approaches. In their proposed solution, a feedback scheme from the number of concurrently running jobs in a key-value based cloud application is used for scaling up the number of allocated physical servers. Since starting servers usually takes longer time than shutting them off, they then pack the incoming work on as few servers as possible and equip each server with a timer. If no requests arrive at an empty server during the timer duration, the server is shut down.

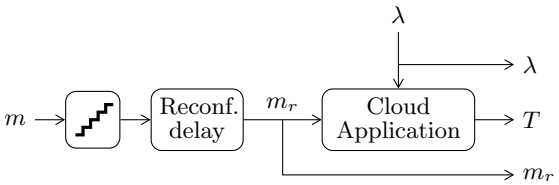


Figure 1: Schematic diagram of the cloud application as a dynamic mapping from desired amount of resources m via deployed resources m_r to the performance metric T. λ is the incoming load of the application and is assumed to measurable. The signal m is also subject to quantization before being sent to the infrastructure.

1.3 Contribution

In this paper, we present an autoscaling solution using inspiration from the Smith predictor. The result is a feedback controller for cloud services that can quickly reconfigure allocated resources when faced with load variations that leads to a lowered QoS. It also avoids the low controller gains and cooldown solutions otherwise commonly used in feedback autoscalers.

In section 2 we present how a cloud application can be seen as a dynamic mapping from resources to a set of performance metrics, and the proposed delay-compensator. In section 3 we focus on a specific case where we apply our proposed solution to control response times. Simulation results from this scenario are shown in section 4. Section 5 concludes the paper.

2. DELAYS IN CLOUD APPLICATIONS

2.1 Dynamic mapping

Cloud applications can generally be regarded as software executing on a set of virtualized resources. Their purpose is often to compute a response to requests made to them. This arrival of requests, usually time-varying in its nature, generates a load on the cloud application, which affects the performance and QoS of a cloud application and can be quantified by a number of relevant metrics, such as response times. In order to keep the performance metrics close to some specific value, as specified by a service level objective (SLO), when facing time-varying load, cloud applications are required to be reconfigurable in terms of resources allocated. We have already outlined how a main challenge for this is the long delays when reconfiguring the deployed amount of resources. Further complicating is the fact that virtual resources usually only can be provisioned in a quantized fashion or are available in preset configurations. For example, the number of VMs provisioned must be integer, memory might only be configured in whole gigabytes, etc.

With this in mind, we view a cloud application as a dynamic mapping from deployed resources and incoming load to a set of performance metrics. This gives us the setup shown in Figure 1. Input is the desired amount of resources

m and outputs are the actual deployed resources m_r, the metric denoted T, and also we assume that we can measure the incoming load λ. The amount of resources also needs quantization before being actuated.

2.2 Delay compensation

The Smith predictor [15] is commonly used for controlling processes with long time delays, and was originally intended for stable, linear, time-invariant SISO systems with a well-known constant time delay. A key assumption for the Smith predictor is the availability of a delay-free model of the system to be controlled. Using this model, the system's response to a given input can be predicted by running a simulation. An identical, but delayed, simulation is also done using the model. Finally, an aggregated measurement signal $\hat{T}$ that adds the output of the real system T and the delay-free model output T_2 and subtracts the delayed model output T_1 can be formed and used for designing a feedback controller. The result is a situation where the feedback only consists of the delay-free model output if the delayed model and system output perfectly matches each other, allowing for higher control gains. Only when there is a mismatch between model and system is the actual system output used for feedback control.

The Smith predictor usually assumes the actuation delays to be constant, which however, as already mentioned, is generally not true for cloud services. For cloud applications, the delays when reconfiguring the deployed resources are stochastic and may even vary during the day [12]. For this reason we modify the original formulation of the Smith predictor so that the delayed model instead uses m_r, the amount of actually deployed resources, as it is not problematic to measure. This gives the setup shown in Figure 2.

As previously mentioned, resources can usually only be deployed in a quantized fashion. But assuming the delay-free model can handle non-quantized amount of resources (m), our setup also comes with the benefit that even changes in m too small to change the output of the quantization actually has an impact on the compensated response time $\hat{T}$ through the delay-free model.

For the remainder of this paper, we focus on applying our solution to a case where we scale the number of homogeneous VMs allocated to a cloud application to ensure that response times are kept bounded. Note that the key assumption in our approach is that we can model the application. Therefore the compensation should be applicable also to other types of resources and applications than the one considered here, such as heterogeneous VMs or MapReduce jobs.

3. RESPONSE TIME CONTROL

In this section we present a case where the delay compensation described in 2.2 is used. The application under consideration is stateless and the VMs are assumed to be homogeneous. A continuous time dynamic model is derived using queueing theory and the feedback loop for controlling the mean response time is closed using a PI controller. For comparison we also implement a threshold-based autoscaler with cooldown based on [1].

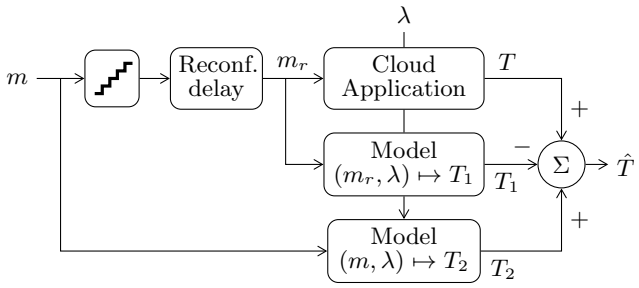


Figure 2: Smith-inspired delay-compensator for cloud applications. The delayed model uses the measured m_r from the cloud application instead using an implementation of a estimate of the delay.

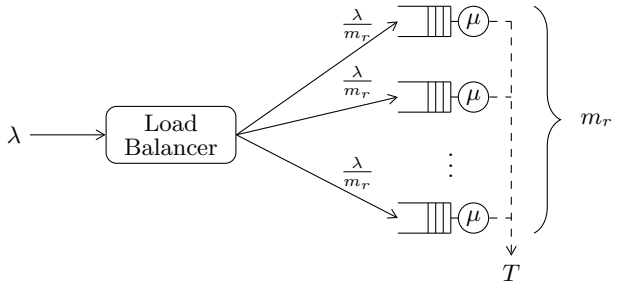


Figure 3: Schematic diagram of the load balancing of m_r running VMs.

3.1 Queueing model

Queueing theory is a commonly used approach for modeling servers. For example, in [7] measurements from web servers were found to be consistent with an `M/G/1` queueing system. In this paper we model each VM as an `M/M/1` queueing system with service rate μ. Traffic is assumed to arrive to the application according to a Poisson process with intensity λ. A load balancer is then used to spread the traffic randomly over m_r currently running VMs, leading to an arrival rate of $\frac{\lambda}{m_r}$ per VM. A schematic diagram of the model is shown in Figure 3. Response times are recorded and sent to the feedback controller, responsible for reconfiguration decisions. Decisions to scale up come with a stochastic startup delay for each VM. Decisions to scale down are effective immediately, as it can be carried out by simply reconfiguring the load balancer and terminating the VM. The quantization effect in this case consists of a ceiling function to make sure that we get the lowest integer value greater than the desired number of VMs.

3.2 Continuous dynamic approximation

Queueing models are generally mostly concerned with the stationary behavior of a system. However in our case, we are also interested in the cloud application dynamics. By viewing the queueing models considered here as systems of flow, we can use the results from [4, 13, 18] to formulate the following approximative model of the dynamics of a `M/M/1` queueing system:

$$\begin{aligned} \dot{x} = f(x, m, \lambda) &= \alpha \left(\frac{\lambda}{m} - \mu \frac{x}{x+1} \right) \\ T = g(x, m, \lambda) &= \mu^{-1}(x+1) \end{aligned} \tag{1}$$

where x corresponds to the queue length, λ/m the arrival rate per running VM, μ the service rate of each VM, T the mean response time and α is a constant used in [13] to better fit the transients of the model to experimental data. It is easy to verify that the equilibrium points of the system (1) for any $0 \leq \lambda < \mu$ coincide with the results from a stationary analysis of a `M/M/1` system. In [16], Tipper *et al* show how system (1) in the case $\alpha = 1$ provides a reasonable approximation to the exact behavior of the non-stationary `M/M/1` queue as found by numerically solving the corresponding Chapman-Kolmogorov equations under certain conditions. Based on the stationary queue length and the stationary response time of the `M/M/1` we can find the output response time T of the flow model.

From now on we will be using the system (1) and its state variable x as the average state of all VMs. Since all virtual machines are equal it is straight-forward to show that

$$\dot{\bar{x}} = \frac{1}{m}\sum_{i=1}^{m} \dot{x} \approx f(\bar{x}, m, \lambda)$$

if we assume all x_i (the states of the individual virtual machine) are the same. This is not true for transients in newly started machines, but as an approximation it is good enough.

Note that system (1) is not dependent on m being integer.

3.3 Control analysis

For control synthesis purposes, we linearize the system equations (1) around the stationary point corresponding to a traffic level λ_0 and response time reference T_{ref}, where we can make use of the fact that stationary queue length x_0 and the stationary number of machines m_0 can be uniquely determined through the other variables as

$$x_0 = T_{ref}\,\mu - 1$$
$$m_0 = \frac{T_{ref}\,\lambda_0}{T_{ref}\,\mu - 1}$$

The linearization yields the following system:

$$\begin{aligned}\Delta\dot{x} =& -\frac{\alpha}{\mu\,{T_{ref}}^2}\Delta x - \alpha\frac{(T_{ref}\,\mu - 1)^2}{{T_{ref}}^2\,\lambda_0}\Delta m \\ &+ \alpha\frac{T_{ref}\,\mu - 1}{T_{ref}\,\lambda_0}\Delta\lambda \\ \Delta T =& \mu^{-1}\Delta x\end{aligned}$$

Note that the dynamics of the linearized system does not change with varying load, while the input gains do. The transfer function from number of machines m to response time T becomes

$$G_p(s) = \frac{\partial g}{\partial x}\left(s - \frac{\partial f}{\partial x}\right)^{-1}\frac{\partial f}{\partial m}\bigg|_{\substack{x=x_0\\ m=m_0\\ \lambda=\lambda_0}} = -\frac{A}{s+a}$$

with $A = \alpha(T_{ref}\,\mu - 1)^2/({T_{ref}}^2\,\lambda_0\,\mu)$ and $a = \alpha/(\mu\,{T_{ref}}^2)$ both greater than zero.

Since the system is of order one, we conclude that a PI controller of the form

$$G_c(s) = K_p + \frac{K_i}{s}$$

should suffice, leading us to the following closed loop dynamics from T_{ref} to T:

$$G_1(s) = \frac{G_c\,G_p}{1 + G_c\,G_p} = \frac{A(K_p s + K_i)}{s^2 + s(a - A\,K_p) - A\,K_i}.$$

The closed loop dynamics from λ to T is given by the transfer function

$$G_2(s) = \frac{G_p}{1 + G_c\,G_p} = -\frac{As}{s^2 + s(a - A\,K_p) - A\,K_i}.$$

We require of the controller that G_1 and G_2 are asymptotically stable. Furthermore we require that the zero in G_1 is not non-minimum phase. Since this zero also shows up in the transfer function from $\Delta\lambda$ to Δm this would otherwise lead to the controller responding to a step increase in traffic by transiently turning off VMs. Lastly, we require that the transfer functions be fully damped, i.e. that all closed loop poles are real. This is because we want to avoid overshoots in the control signal when faced with a step shaped disturbance or reference change, as it would lead us to starting up VMs that are almost immediately turned off again. Combining these requirements puts the following constraints on the controller parameters:

$$\begin{aligned}K_i &< 0\\ K_p &\leq 0\\ -4AK_i &\leq (a - AK_p)^2\end{aligned}$$

In order to simplify controller design, we can reparameterize the closed loop poles in the following way:

$$s = -\frac{a - AK_p}{2} \pm \sqrt{\frac{(a - AK_p)^2}{4} + AK_i} = -\varphi \pm \xi,\ \varphi \geq \xi \geq 0$$

allowing us to find the following expression for the controller parameters:

$$\begin{aligned}K_p &= \frac{a - 2\varphi}{A},\ \varphi \geq \frac{a}{2}\\ K_i &= \frac{\xi^2 - \varphi^2}{A}\end{aligned}$$

where the condition on φ makes sure that the zero in $G_1(s)$ is minimum phase.

3.4 Threshold-based controller

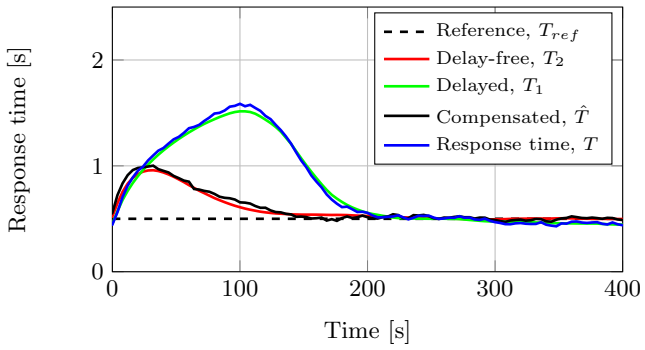
For comparison we also implement a threshold-based controller with cooldown, based on the autoscaling solution used in Amazon Web Services [1]. The controller measures the average response times over a time period h, and compares it to two given thresholds, one upper T_{upper} and one lower T_{lower}. Whenever h_t measurements in a row are either above the upper or below the lower threshold, an autoscaling event is triggered, either trying to start or shut down one VM.

Successfully executing an autoscaling event (shutting down or starting up a VM) also starts a cooldown period, with length $h_{cooldown}$. Whenever a cooldown period is running no new autoscaling events are triggered.

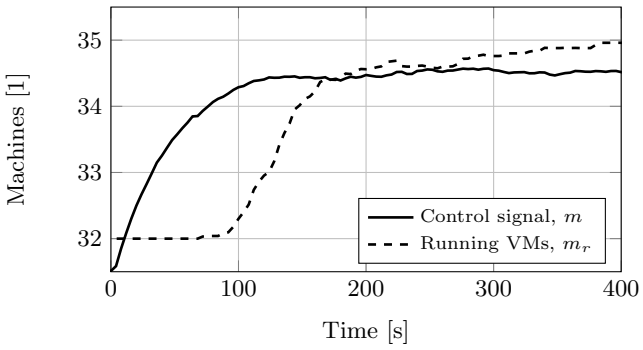
4. EXPERIMENTAL RESULTS

4.1 Delay-compensated control

To evaluate the delay-compensator described in Section 2.2 we run a set of discrete event-based simulation experiments.

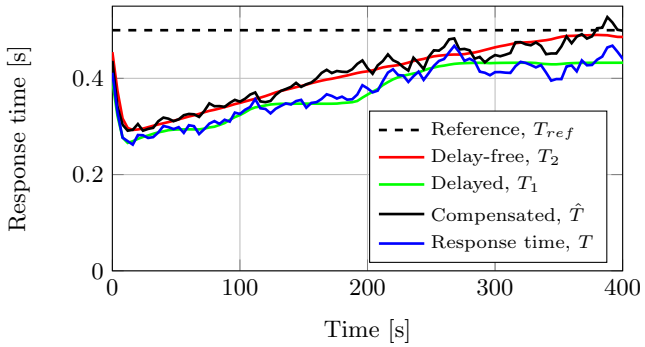


(a) Response time results from simulation of step up. The compensated response times reach the reference much before the actual response times.

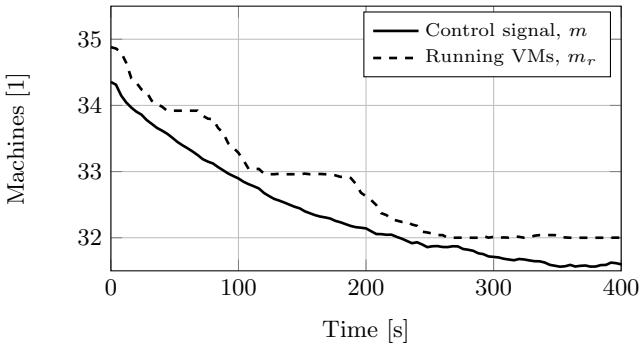


(b) Control signals from simulation of step up. The controller manages to respond to the change in load with little overshoot, which is important.

Figure 4: Results from simulating a step-shaped increase in traffic.



(a) Response time results from simulation of step down. The difference between delayed and delay-free is that the delay-free model has no quantization.



(b) Control signals from simulation of step down. The controller gradually turns off machines to find the equilibrium.

Figure 5: Results from simulating a step-shaped decrease in traffic.

The cloud application is an implementation of the model described in Section 3.1. The PI controller derived in section 3.3 is implemented in discrete time as such:

$$
\begin{aligned}
e_k &= T_{ref} - \hat{T}_k \\
i_k &= i_{k-1} + K_i\, h\, e_k \\
m_k &= K_p\, e_k + i_k
\end{aligned}
$$

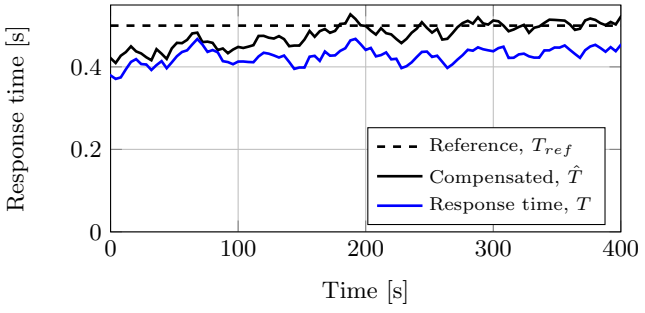
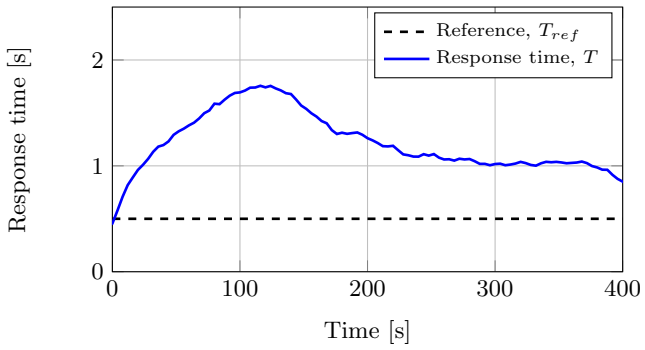


Figure 6: Steady state with $\lambda = 630$. The controller finds the lowest number of machines to come below T_{ref} and then compensates for the difference.

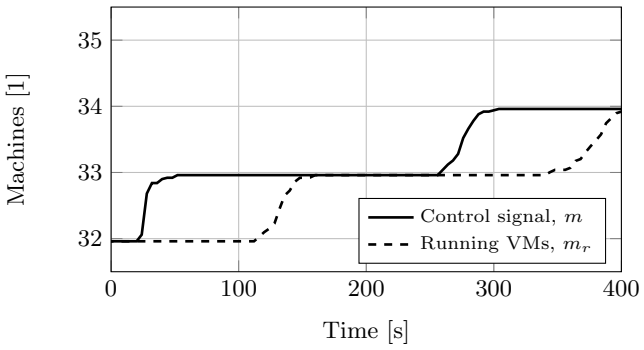
where m_k is the control signal, i_k is the integrator state and $\hat{T}_k$ is the mean of all delay-compensated response times between sampling points $k-1$ and k. For this implementation we omit anti-windup since the only saturation in the system is $m > 0$, and all experiments are designed to stay far away from that point. The VMs have a service rate $\mu = 22$ and uniformly distributed startup delays in the interval $[80, 120]$ seconds, while shutting down a VM is immediate. The linearization point is chosen as $\lambda_0 = 630$ and $T_{ref} = 0.5$ s, and the controller parameters are chosen so that $\varphi = 0.0545$, $\xi = 0.0432$. The controller runs every $h = 2$ s. Experimental trial showed that using $\alpha = 0.5$ in our cases provided a reasonable transient fit.

The delay compensator updates the state of the delayed and the delay-free model on every request leaving the cloud application. The continuous models are discretized using the Runge-Kutta method.

In the first experiment, the incoming traffic to the application is changed as a step from 630 to 690 requests per second. We perform a set of 25 step response experiments, and aggregate the results to calculate the average response times and number of VMs over a window of 4 seconds. The results are shown in Figure 4.



(a) Response times for the step up scenario when using the threshold controller with cooldown



(b) Number of machines for the step up scenario when using the threshold controller with cooldown

Figure 7: Results from simulating a step-shaped increase in traffic for the threshold-based controller.

As we can see in Figure 4a the real response times reach its highest point about the same time as the first newly started VM becomes active. Figure 4b shows the average control signal (m) and running VMs (m_r). The controller manages to respond to the change in load, without significant overshoot, which is the typical problem caused by actuation delays.

Plots of simulations of the step down from 690 to 630 per second is shown in Figure 5. The difference between delayed and delay-free model while scaling down is that the delay-free model has no quantization. In less than 300 seconds we reach the theoretical stationary value $m_r = 32$.

Shown in Figure 6 is a plot of the average behavior when the system is approaching steady state with $\lambda = 630$. As can be seen, response times are not varying around T_{ref}, but slightly below. This is because $m_0 = T_{ref}\,\lambda_0/(T_{ref}\,\mu - 1) = 31.5$ is not an integer. Since we can only run integer number of machines and the ideal number is a fraction, an uncompensated PI controller would oscillate between the two values 31 and 32 for m_r. The compensated controller on the other hand finds the smallest integer m_r larger than m_0 and compensates away the part of the error that can not be removed without exceeding T_{ref}. T approaches $T_0 = \mu^{-1}(\frac{\lambda_0}{\mu\lceil m_0\rceil - \lambda_0} + 1) \approx 0.43$ s instead of $T_{ref} = 0.5$ s.

With this controller, for all 25 experiments, we use on average 33.7 machine hours per hour. The mean response time during scale-up is 0.804 seconds and during scale-down 0.373 seconds.

4.2 Threshold-based controller

For comparison we also run the same experiment as previously described with the threshold controller described in 3.4. The controller is run with the parameters $T_{lower} = 0.35$ s, $T_{upper} = 0.6$ s, $h_t = \frac{20\text{ s}}{h}$, $h_{cooldown} = 120$ s.

The mean response times and number of running VMs are shown in Figure 7 respectively. As we can see the controller does not even manage to get the response times back to the reference value before 400 seconds have passed. Due to the fact that the controller cannot act while in a cooldown period, we respond too slowly to the increase in traffic.

With this controller, for the full experiment, we use 33.3 machine hours per hour. Mean response time during scale-up is 1.224 seconds and during scale-down 0.327 seconds.

4.3 Discussion

As can be seen in Figures 4 and 7 the delay-compensated controller manages to quickly respond to changes in the incoming load. The control signal m reaches its final value of $34 < m < 35$ before the first actual machine has even started. Since the threshold controller needs to wait for its cooldown to pass it is slow to respond. This is also why the delay-compensated controller uses more resources on average.

In Figure 6 we see how we are left with a stationary offset between the response times T and T_{ref}. Since no integer number of virtual machines will result in stationary response times at T_{ref}, the controller finds the lowest amount of machines needed to stay below T_{ref} and then compensates away the error which can't be controlled away.

5. CONCLUSIONS

In this paper we have extended the, in the control community, commonly used Smith predictor for compensating for VM startup delay. The classic Smith predictor needs knowledge about the length of the time delay, but since it is reasonable to assume that we can at all times know the number of currently running VMs we don't need to know or implement the delay. The only thing we need is a model of the behavior of the cloud application after the delay.

Through simulations we show that the compensator can compensate for the startup delay of VMs and that the resource management can be solved using a simple PI controller. Thanks to the delay-compensation the controller can reach the final number of machines before the first machine has even started. The compensator picks the lowest number of VMs which gives response times below the reference.

6. ACKNOWLEDGMENTS

This work was supported by the Swedish Research Council (VR) for the project "Cloud Control", and through the LCCC Linnaeus and ELLIIT Excellence Centers.

7. REFERENCES

[1] Auto scaling concepts — Amazon Web Services documentation. `https://web.archive.org/web/20140729191545/http://docs.aws.amazon.com/AutoScaling/latest/DeveloperGuide/AS_Concepts.html`. Accessed: 2014-08-27.

[2] Google compute engine autoscaler — Google Cloud Platform Documentation. `https://web.archive.org/web/20141201094332/https://cloud.google.com/compute/docs/autoscaler/`. Accessed: 2014-12-01.

[3] How auto scale cooldowns work — Rackspace Knowledge Center. `https://web.archive.org/web/20141117122211/http://www.rackspace.com/knowledge_center/article/how-auto-scale-cooldowns-work`. Accessed: 2014-11-17.

[4] Carson E Agnew. Dynamic modeling and control of congestion-prone systems. *Operations research*, 24(3):400–419, 1976.

[5] Ahmed Ali-Eldin, Johan Tordsson, and Erik Elmroth. An adaptive hybrid elasticity controller for cloud infrastructures. In *Network Operations and Management Symposium (NOMS), 2012 IEEE*, pages 204–212. IEEE, 2012.

[6] Mihaly Berekmery, Damian Serrano, Sara Bouchenak, Nicolas Marchand, Bogdan Robu, et al. A control approach for performance of big data systems. 2014.

[7] Jianhua Cao, Mikael Andersson, Christian Nyberg, and Maria Kihl. Web server performance modeling using an M/G/1/K* PS queue. In *Telecommunications, 2003. ICT 2003. 10th International Conference on*, volume 2, pages 1501–1506. IEEE, 2003.

[8] Anshul Gandhi, Mor Harchol-Balter, Ram Raghunathan, and Michael A Kozuch. Autoscale: Dynamic, robust capacity management for multi-tier data centers. *ACM Transactions on Computer Systems (TOCS)*, 30(4):14, 2012.

[9] Zhenhuan Gong, Xiaohui Gu, and John Wilkes. Press: Predictive elastic resource scaling for cloud systems. In *Network and Service Management (CNSM), 2010 International Conference on*, pages 9–16. IEEE, 2010.

[10] Harold C Lim, Shivnath Babu, Jeffrey S Chase, and Sujay S Parekh. Automated control in cloud computing: challenges and opportunities. In *Proceedings of the 1st workshop on Automated control for datacenters and clouds*, pages 13–18. ACM, 2009.

[11] Tania Lorido-Botrán, José Miguel-Alonso, and Jose Antonio Lozano. Auto-scaling techniques for elastic applications in cloud environments. *Department of Computer Architecture and Technology, University of Basque Country, Tech. Rep. EHU-KAT-IK-09*, 12:2012, 2012.

[12] Ming Mao and Marty Humphrey. A performance study on the VM startup time in the cloud. In *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, pages 423–430. IEEE, 2012.

[13] Kenneth Lloyd Rider. A simple approximation to the average queue size in the time-dependent M/M/1 queue. *Journal of the ACM (JACM)*, 23(2):361–367, 1976.

[14] Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, page 5. ACM, 2011.

[15] Otto J M Smith. Closer control of loops with dead time. In *Chem. Eng. Progr.*, volume 53, pages 217–219, 1957.

[16] David Tipper and Malur K Sundareshan. Numerical methods for modeling computer networks under nonstationary conditions. *Selected Areas in Communications, IEEE Journal on*, 8(9):1682–1695, 1990.

[17] Bhuvan Urgaonkar, Prashant Shenoy, Abhishek Chandra, Pawan Goyal, and Timothy Wood. Agile dynamic provisioning of multi-tier internet applications. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 3(1):1, 2008.

[18] Wei-Ping Wang, David Tipper, and Sujata Banerjee. A simple approximation for modeling nonstationary queues. In *INFOCOM'96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, volume 1, pages 255–262. IEEE, 1996.