Control strategies for predictable brownouts in cloud computing

Martina Maggio * **Cristian Klein** ** **Karl-Erik Årzén** *

* *Department of Automatic Control, Lund University*
** *Department of Computing Science, Umeå University*

Abstract: Cloud computing is an application hosting model providing the illusion of infinite computing power. However, even the largest datacenters have finite computing capacity, thus cloud infrastructures have experienced overload due to overbooking or transient failures.
The topic of this paper is the comparison of different control strategies to mitigate overload for datacenters, that assume that the running cloud applications are cooperative and help the infrastructure in recovering from critical events. Specifically, the paper investigates the behavior of different controllers when they have to keep the average response time of a cloud application below a certain threshold by acting on the probability of serving requests with *optional computations* disabled, where the pressure exerted by each request on the infrastructure is diminished, at the expense of user experience.

Keywords: Computer systems, Feedback loops, Model-based control, Multiprocessor systems, Probabilitstic models, Queuing theory.

1. INTRODUCTION

Cloud computing is changing the way computing resources are provisioned. In the past, it was necessary to buy a new physical machine every time the hosted application was not reaching its target performance. Now, on the contrary, it is possible to simply upgrade the virtual machine that is deployed in the cloud infrastructure, letting the cloud provider decide when to physically upgrade the infrastructure. This decoupling from the infrastructure allows cloud applications to be scaled on-demand, for example as needed when the number of users increases because the application becomes popular. However, it comes with the inherent assumption that the cloud provider has infinite computing capacity that can be delivered instantaneously.

Given the tendency to move as many applications as possible to the cloud, this assumption will eventually reach its limits. Building datacenters with the necessary computing power to serve all the incoming requests, despite unexpected events, will become prohibitive. Therefore, we need to start managing our cloud applications in a smarter way, letting them take into account infrastructure limitations and decrease the burden they impose on the datacenter as required to recover from overload due to hardware failures (Guo et al., 2013), overbooking or other emergency situations.

This work originates from the observation that these emergency situations might require drastic countermeasures. In a typical situation, overloads have the direct consequence of application unavailability, i.e., a blackout. This paper argues that it would better serve the user to keep the application available, degrading user experience instead, what we call a *brownout*.

Brownout builds on the basic mechanism called *optional computations*. A cloud application typically serves requests providing responses. These responses can be decomposed into computations, some of which are necessary to satisfy the user's request, others are optional, merely improving user experience. This paper's proposal is to probabilistically turn off the invocation of the components that produce these optional parts of the response to deal with overload conditions predictably. The solution leverages control theory to synthesize the adaptation mechanism that selects the probability of serving a request with optional components activated or deactivated, based on measurements obtained from the running application. In Klein et al. (2014) we proposed a simple controller that keeps the average user perceived latency, also called response time, below a certain threshold. The purpose of this paper is to compare different control solutions to improve on the simple controller.

The contribution of this paper lays in the analysis of different control strategies for brownout-compliant applications. The paper compares the behavior of different controllers when the cloud application and the infrastructure hosting it is exposed to variations. These variations include sudden increases in the popularity of the cloud application, through the number of users accessing it. Also, it compares how the controllers react to changes in the amount of resources allocated to the cloud application. It finally tests the response of the application to requirement changes, for example due to more challenging — possibly transient — service requirements.

⋆ This work was partially supported by the Swedish Research Council (VR) for the projects "Cloud Control" and "Power and temperature control for large-scale computing infrastructures" and through the LCCC Linnaeus Center. Also, we received partial support from the ELLIIT Excellence Center.

The remainder of this paper is organized as follows. Section 2 describes the model of the cloud application. Section 3 shows the possible control strategies that the paper studies, while Section 4 describes the experimental evaluation carried on to compare the controllers. We simulated the different control strategies, demonstrating their advantages in specific conditions. The paper also discusses the related literature in Section 5 and draws some conclusions in Section 6.

2. THE MODEL OF THE CLOUD APPLICATION

Cloud applications serve multiple users and their computations can generally be seen as independent and stateless patterns of requests and responses, as described by Fielding and Taylor (2002). An essential requirement of these applications is that they should act in a time-sensitive way, otherwise unsatisfied users would abandon the service. Nah (2004) showed that a tolerable waiting-time for the users is between two and four seconds.

Every response that is produced can be divided into two different parts: the necessary information to be given to the user and some optional components that improve user experience but do not constitute the main reason why the user queried the application. For example, when browsing items on an e-commerce website, showing recommendations about similar items to the user greatly improves the navigation, but is resource hungry for the underlying infrastructure. Denoting τ_m the time needed to execute the mandatory part of the response generation and with τ_o the optional computation time, the response time r_i for each request can be written as

$$r_i = \tau_m + \eta_i \cdot \tau_o \tag{1}$$

where η_i is either one or zero and denotes if the optional computation has been performed or not, respectively. We select as our control variable the probability $\pi(k)$ of executing optional computations between the controller execution at time k and the next execution at time $k+1$. Every time a request is processed, a Bernoulli trial is performed, to determine if the optional computations should be executed, based on the current value of π.

Equation (1) can be used to determine the response time of the cloud application, otherwise modeled as an $M/D/n$ queue, where n workers serve requests that takes a deterministic (D) time to be processed. The requests arrival rates are determined by a Poisson process (M). This queue is memoryless and every request is independent, with a determined response time. n represents the number of independent threads waiting for incoming requests.

Equation (1) represents the behavior of the application. However, it does not define a model that is suitable for control purposes. In fact, it does capture only the behavior of each single request. On the contrary, the M/D/n queue captures the behavior of the application in steady state, but cannot be used to synthesize a control strategy, because it does not describe the transient phase. This paper tries to overcome the mentioned limitation using a primitive, yet useful, model. We assume that the average response time of the web application, measured at regular time intervals, follows the equation

$$t(k+1) = \alpha(k) \cdot \pi(k) + \delta t(k) \tag{2}$$

i.e., the average response time $t(k+1)$ of all the requests that are served between time k and time $k+1$ depends on a time varying unknown parameter $\alpha(k)$ and can have some disturbance $\delta t(k)$ that is *a priori* non-measurable. $\alpha(k)$ takes into account how our control value — the probability of executing the optional computations — affects the response time. $\delta t(k)$ is an additive correction term that models exogeneous variations, like a variation in retrieval time of data due to cache hit or miss. The controller design should aim at canceling the disturbance $\delta t(k)$ and selecting the value of $\pi(k)$ so that the average response time would be equal to our setpoint value.

3. CONTROL STRATEGIES

In this section, we discuss different control alternatives, designed using the model defined by Equation (2). It also motivates why each of these control strategies might be considered a good idea for the specific problem. The controllers measure the average response time — or average lantecy — of the cloud application and select the probability π of executing the optional code. This section discusses seven control alternatives. The controller class spans from PIs and PIDs to feedforward plus feedback and deadbeat regulators. The estimation method can be a Recursive Least Square (RLS) filter or the simple application of Equation (2). In the following this second estimation strategy is called “bare” estimation methodology.

Adaptive PI controller: The first proposed alternative is synthesized through loopshaping, constraining the transfer function of the closed loop. As a first step of the design, $\alpha(k)$ is treated as a known parameter and assumed to be constant and equal to α. In a second phase, the current value of $\alpha(k)$ is estimated, obtaining therefore an adaptive controller. The transfer function $P(z)$ from the input π to the measured average response time is

$$P(z) = \frac{T(z)}{\pi(z)} = \frac{\alpha}{z} \tag{3}$$

where $T(z)$ is the Z-transform of $t(k)$ and $\pi(z)$ transforms $\pi(k)$.

The control system is designed so that the closed loop transfer function $G(z)$ is equal to

$$G(z) = \frac{C(z) \cdot P(z)}{1 + C(z) \cdot P(z)} = \frac{1 - p_1}{z - p_1} \tag{4}$$

where p_1 is a stable pole, in the experiments 0.9. Substituting the plant transfer function of 3 into 4 it is possible to derive the expression $C(z) = \frac{(1-p_1) \cdot z}{\alpha(z-1)}$ for the controller, which is a PI controller where the controller coefficients depend on α. Applying the inverse Z transform on $C(z)$,

$$\pi(k+1) = \pi(k) + \frac{1 - p_1}{\alpha} \cdot e(k+1) \tag{5}$$

where $e(k+1)$ is error between the average latency and its setpoint. This solution is coupled with two possible online estimation methods for $\alpha(k)$. In the first case, we simply invert the relationship given by Equation (2), substituting the measured average latency and the value of the previous control variable and computing a measured α. In the second case, an RLS estimator is implemented to keep track of variations of α, using a forgetting factor of 0.2.

Adaptive deadbeat controller: Since the plant pole is inside the unit circle, an adaptive deadbeat controller can be synthesized with the same procedure used to derive the adaptive PI, imposing that the pole p_1 lays in zero. A deadbeat controller may be a good choice, since the controller should react faster in case of sudden changes that the plant might have, as happens with flash-crowds or hardware failures.

Adaptive PID controller: It is also interesting to test a PID controller, adding a derivative action to the adaptive PI controller. In fact, the derivative action can increase stability and improve settling time thanks to its future trend prediction. The derivative gain is manually tuned, selecting the value of 0.2.

Feedforward plus feedback controller: Although it may require significant engineering effort, it is possible to obtain the number of currently queued request from the application, therefore it is interesting to test if exploiting this information could result in a better controller. Denoting $m(k)$ the number of queued request at time k, we can predict that the latency experienced by the i-th request that entered the queue is $(\tau_m + \pi \cdot \tau_o) \cdot i$, therefore the average latency experienced by the m requests should be

$$(\tau_m + \pi \cdot \tau_o) \cdot \frac{m+1}{2}. \tag{6}$$

This value can be used to setup a feedforward control strategy, that selects

$$\pi(k+1) = (1-\mu) \cdot \pi(k) + \mu \cdot \frac{2 \cdot \bar{t}}{\tau_o \cdot [m(k)+1]} - \frac{\tau_m}{\tau_o} \tag{7}$$

where $\bar{t}$ is the desired latency value and μ is a discount factor. The implementation uses the value of 0.9 for the discount factor. The feedforward controller is also coupled with a feedback PID, with proportional gain 0.15, and integral and derivative gain 0.1. These values are found with empirical tests and resulted to be optimal for the specific control problem. Also, the feedfoward regulator is executed with a period that is 10 times the period of the feedback one. One of the main differences between this controller and the previous ones is that the other alternatives were estimating the model parameter α, while in this case the controller receives more information from the infrastructure but does not perform online estimation.

4. EXPERIMENTAL VALIDATION

This section discusses our experimental validation. First it describes the SimEvents (Clune et al., 2006) based simulator that was built in Simulink to simulate the behavior of brownout-compliant application and the action of different control strategies. Then, it illustrates the comparison metrics that will be used for two experiments and their results. The first experiment is carried on varying the number of users accessing to the application and the application requirements in terms of the desired average response time. The second experiment tests the amount of resources given to the application in terms of the amount of CPUs that the application can use during the simulation.

4.1 Brownout-compliant simulator

SimEvents (Clune et al., 2006) is a well known discrete event simulation tool. In this paper, it is used to implement a model of a brownout-compliant cloud application. Here, we introduce both the simulator and the parameters that are not changed during the experiments. The parameters that specifically belongs to one experiment are introduced in the corresponding section.

The SimEvent based model is simulating both the infrastructure and the cloud application that is deployed on top of it. The infrastructure is represented as n servers, where n is the number of CPUs that the application is allocated. The amount of available resources is decided by the cloud provider, therefore the simulation platform is built to be able to dynamically change this value during the simulation execution. A request generator is included in the simulation, that is able to load our server with different input rates. It is possible to simulate the variation of the input rate in order to test the application's reaction in conditions like flash-crowds. In the experiments, we vary the amount of requests introduced changing the average of an exponential distribution. By attaching a tag to every request when it enters the system, it is possible to obtain statistics on the average latency and to provide the necessary measurements for the control system to operate.

The cloud application simulates the service of a request by generating a random number and comparing it to the control signal π and deciding if the optional part should be enabled or disabled. Plausible values for the optional computation τ_o time and the mandatory computation time τ_m were selected, but our simulator also introduces some variability to represent the real server conditions — where caching effects, locks and shared resources could introduce noise. Therefore the simulator uses random numbers coming from a normal distribution having a variance of 0.05. The average for the optional execution time is 0.3 seconds, while for the mandatory part of the request we assume that on average 0.1 seconds are consumed. The simulator allows the user also to vary the desired average latency. Notice that the chosen values are simply potential numbers for the simulations. Clearly, the infrastructure should be sized to serve the number of incoming requests. Usually, sizing the computing infrastructure is done offline, deciding the amount of servers to allocate to a certain application with a prediction of the incoming load and of the service times. It can also be done dynamically by changing the number of CPUs assigned to the virtual machine hosting a cloud application. The adaptation of the number of CPUs is not considered in this work, instead, the amount of received resources to execute on is seen as an exogenous disturbance and the control solutions should react to it by selecting the best possible value of π.

Finally, the simulations are run for 1000 seconds and the controllers acts every 5 seconds, using as feedback signal the average latency measured over the last 15 seconds of execution. The simulator also selects the seed for the random number generation so that different experiments are comparable and repeatable.

4.2 Comparison metrics

To compare the behavior of the different controllers five different metrics are used. Four of them came from the control theoretical domain, while the last one is more cloud oriented. The control-based metrics are the Integral of

Table 1. Performance metrics for the experiment varying application popularity and requirements.

Controller	$opt_{\%}$	ISE	ISTE	IAE	ITAE
Adaptive PI (bare)	0.3005383	4535.517	1904275	5374.772	2203108
Adaptive PI (RLS)	0.3014424	4117.492	1534573	5168.061	2002305
Adaptive PID (bare)	0.3036972	2523.083	767550	4191.676	1562685
Adaptive PID (RLS)	0.2912218	2214.995	768936	3993.482	1582531
Adaptive deadbeat (bare)	0.2334130	15527.860	5322117	10590.770	3886003
Adaptive deadbeat (RLS)	0.2658776	12437.960	4663479	9613.271	3793741
Feedforward plus feedback	0.2723032	1566.729	498571	3707.945	1384026

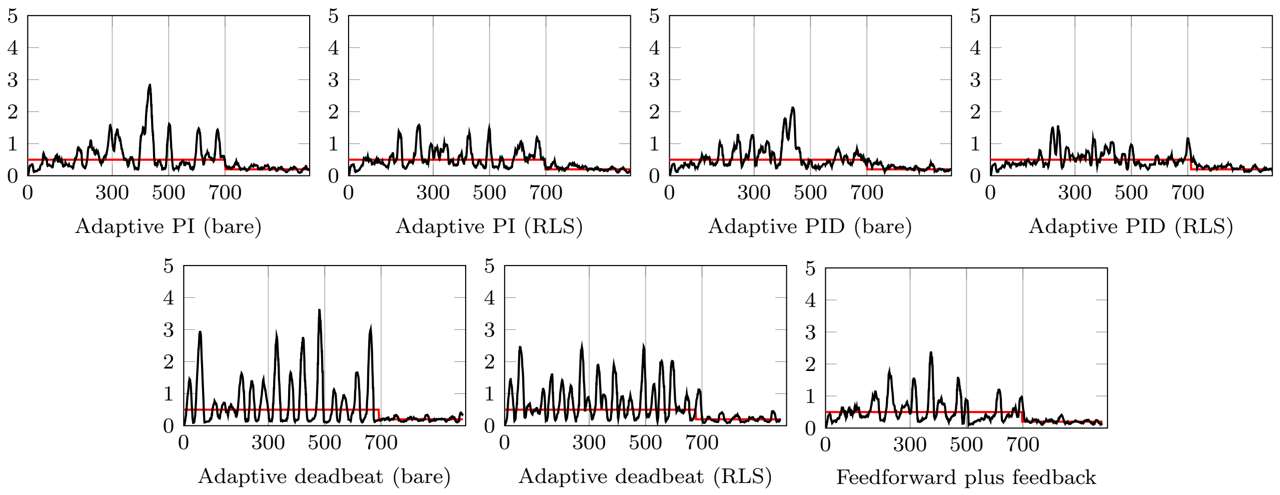

Fig. 1. Average latency (black) and corresponding setpoint (red) for the experiment varying application popularity and requirements.

the Squared Error (ISE), the Integral of Timed Squared Error (ITSE), the Integral of the Absolute Error (IAE) and the Integral of Timed Absolute Error (ITAE). They are calculated as follows.

$$ISE = \sum [e(t)]^2 \tag{8}$$

$$ISTE = \sum t \cdot [e(t)]^2 \tag{9}$$

$$IAE = \sum |e(t)| \tag{10}$$

$$ITAE = \sum t \cdot |e(t)| \tag{11}$$

To evaluate the control strategy also from a cloud perspective, we use the percentage of requests served with the optional code enabled, $opt_{\%}$. Clearly, the application developer would like the optional code to be enabled as often as possible, increasing revenue. Therefore, this is an important parameter to be optimized and it is a good index for comparison. Intuitively, the best controller achieves the highest value of $opt_{\%}$ while keeping low error-based metrics.

4.3 Varying application popularity and requirements

This experiment tests the effect of application popularity and of the average response time requirement set for the cloud service. To do so, we launch a server with one single CPU and do not change the resources available to the application during the simulation. To simulate the change in the application popularity the simulation is started with 3 requests per second to be served by the web application and the number of requests per second is increased to 5 after 300 seconds from the simulation start. At time 500 the load is reduced back to the initial value. To simulate the requirement change, the desired average latency is set at 0.5 seconds per request and decreased to 0.2 seconds at time 700.

The system is tested with all the seven control strategies introduced and the performance metrics described in Section 4.2 are computed, reporting the values in Table 1. Figure 1 shows the average latency and the setpoint value during the experiment with the seven different control strategies. From the reported numbers, it is evident that the control strategies resulting in the highest number of optional code served are the Adaptive PIs and PIDs. It is also clear that adding a derivative action improves the control performances, the errors are lower. The adaptive PID with RLS estimator is able to keep the ISE lower than the others, at the expense of fewer recommendations served. The deadbeat controllers, although being very fast in reacting to changes, are also too sensitive to disturbances and therefore perform poorly compared to their adaptive counterparts — both the percentage of requests served with optional code enabled and the control based metrics are worse than their counterparts.

On the contrary, the feedforward plus feedback controller serves less optional code (27% of requests are served with the optional code enabled, compared with the 30% achievable with adaptive control strategies) but reduce the error (all the control indexes are better than the respective counterparts). From this experiment it is possible to conclude that the feedforward plus feedback controller is

Table 2. Performance metrics for the experiment varying resource availability.

Controller	$opt_{\%}$	ISE	ISTE	IAE	ITAE
Adaptive PI (bare)	0.7148029	793.073	314201	3115.364	1385175
Adaptive PI (RLS)	0.7183862	1684.972	593336	3421.560	1484901
Adaptive PID (bare)	0.7223227	1211.365	439000	3095.480	1350509
Adaptive PID (RLS)	0.7173301	645.596	240696	2677.837	1209246
Adaptive deadbeat (bare)	0.7114378	1279.612	591899	4097.543	1873055
Adaptive deadbeat (RLS)	0.7103646	1192.516	628194	3810.441	1954958
Feedforward plus feedback	0.6691487	780.614	296312	3287.870	1391454

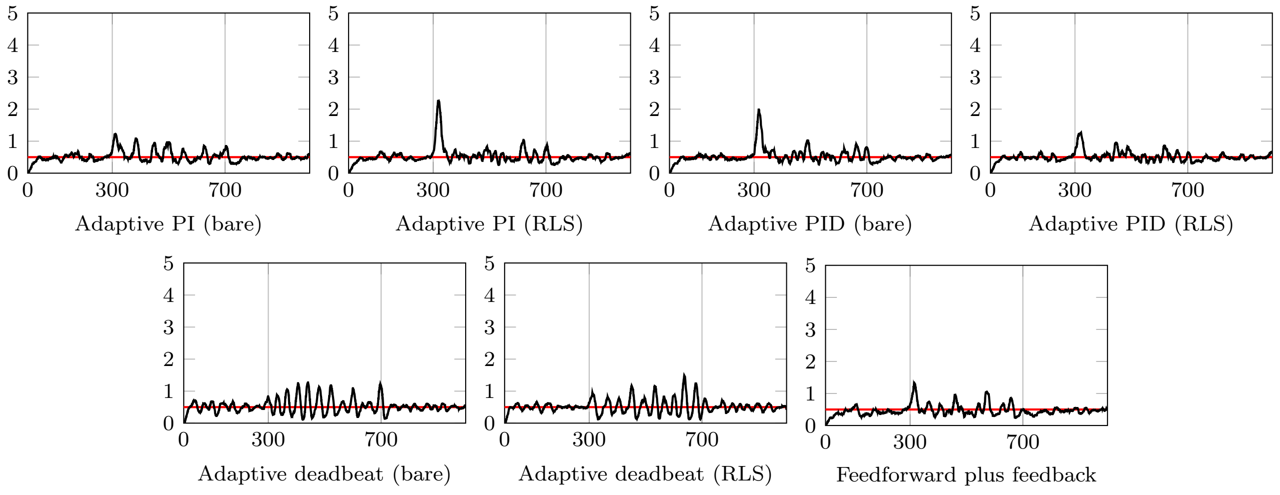

Fig. 2. Average latency and corresponding setpoint for the experiment varying resource availability.

more conservative and exploits the additional information received from the cloud application to better react to disturbances and track the setpoint. The adaptive PID controllers (especially with the RLS estimator) still achieve a low error, serving more optional code. Hence, if the service level objectives are very strict and variations of the cloud application behavior are subject to a high penalty, it is advisable to implement the feedforward plus feedback strategy, while in case the requirements are less strict, the adaptive PID controllers are a very good compromise between control performances and user experience.

4.4 Varying resource availability

This experiment tests the effect of resource availability changes on the cloud service. The simulation is launched assuming that the application has four CPUs to be used. At time 300 two CPUs are removed from the application, e.g. due to a hardware failure, and added back at time 700. During the interval between 300 and 700, the application can use only two CPUs. It should therefore decrease the number of requests served with optional code, still maintaining the average latency close to the setpoint value. The experiment simulates the arrival of 5 requests per second, which can be served almost entirely with the optional code enabled when four CPUs are available and should be served with about half of the optional code enabled when only two CPUs are available.

The computed performance metrics for this experiment are reported in Table 2. Figure 2 shows the average latency and the setpoint value during the experiment execution with the seven different control strategies.

The results confirm the insight obtained with the previous experiment. The feedforward plus feedback controller serves slightly less recommendations, however, lowering the error compared to the other controllers. In this case, the adaptive PID tuned with the bare estimator is able to obtain a lower ISE compared to the feedforward plus feedback controller. However, the time-weighted errors are higher for this controller with respect to the feedforward plus feedback case. Also in this case, the deadbeat controllers are too sensitive to model perturbation and do not offer any significant advantage over the other strategies.

Overall, the results show that, if the effort of obtaining the necessary information to build the feedforward plus feedback controller is affordable, it is worth implementing this control strategy. Otherwise, an adaptive PID strategy can be used without sacrificing control performance.

5. RELATED WORK

This paper discusses control strategies for building self-adaptive cloud applications through brownout. In software engineering, self-adaptivity is playing a key role in the development of software systems (Kramer and Magee, 2007; Cheng et al., 2009; Kephart, 2005) and control theory has proved to be a useful tool to introduce adaptation in such systems (Diao et al., 2006; Weyns et al., 2012; Filieri et al., 2011; Brun et al., 2009). Many attempts have been made to apply control theory to computing systems, as the survey by Patikirikorala et al. (2012) testifies. However, the research is still in a preliminary stage and the achievable benefits are yet to be clearly defined (Zhu et al., 2009; Hellerstein, 2010). The presented problem and solution were inspired by the idea that there might be multiple code alternatives for the same functionality (Diniz and Rinard, 1997; Ansel et al., 2009) and not all the code that a software application is executing is necessary. Often, some of the application code might be skipped to achieve better performance, for example as implemented by Misailovic et al. (2010). A similar concept has been proposed in the web context. Degrading the static content of a website was first proposed by Abdelzaher and Bhatti (1999) and has been subsequently extended to dynamic content by Philippe et al. (2010). However the idea has not yet been applied to cloud computing applications.

The self-adaptive cloud application that is closest to the vision proposed in this paper is Harmony (Chihoub et al., 2012). It adjusts the consistency-level of a distributed database as a function of the incoming end-user requests, so as to minimize resource consumption. This is a specific example of how a cloud application can be compliant with the proposed paradigm. This paper however presented a general technique to introduce control theory into a vast class of cloud applications and a comparison of the results of different control solutions. In the current research on self-adaptivity, usually one single control strategy is designed and refined until it matches specific requirements. However, an in-depth exploration of different control strategies and trade-offs is rarely provided. Thus, it is unknown whether a different control strategy could have offered a better solution for the specific problem. Our works fills this gap, by offering a comparison of different strategies for the same control problem.

6. CONCLUSION

This paper proposed different control strategies for cloud computing applications that are made self-adaptive by exposing optional computations and allowing them to be skipped on a per-request basis. It defined a model for the application and validated control strategies by simulating different load conditions with SimEvents, a Matlab toolbox to simulate discrete event systems. The simulator includes the typical variations that a cloud application can be subject to, from resource availability to popularity and requirement changes. The paper describes two case studies, involving the various *stimuli* that our simulator allows us to give to the cloud application.
The obtained results indicate that either the controllers need information about the plant status — the number of queued requests and an estimation of the mandatory and optional time, for the feedforward plus feedback case — or they should resort to online estimation. If this information is easy to obtain, the feedforward plus feedback strategy give significant improvements.

REFERENCES

Abdelzaher, T.F. and Bhatti, N. (1999). Web content adaptation to improve server overload behavior. In *WWW*, 1563–1577.

Ansel, J., Chan, C., Wong, Y.L., Olszewski, M., Zhao, Q., Edelman, A., and Amarasinghe, S. (2009). Petabricks: A language and compiler for algorithmic choice. In *PLDI*.

Brun, Y., Marzo Serugendo, G., Gacek, C., Giese, H., Kienle, H., Litoiu, M., Müller, H., Pezzè, M., and Shaw, M. (2009). Software engineering for self-adaptive systems. 48–70.

Cheng, B., de Lemos, R., Giese, H., Inverardi, P., Magee, J., Andersson, J., Becker, B., Bencomo, N., Brun, Y., Cukic, B., Di Marzo Serugendo, G., Dustdar, S., Finkelstein, A., Gacek, C., Geihs, K., Grassi, V., Karsai, G., Kienle, H., Kramer, J., Litoiu, M., Malek, S., Mirandola, R., Müller, H., Park, S., Shaw, M., Tichy, M., Tivoli, M., Weyns, D., and Whittle, J. (2009). Software engineering for self-adaptive systems: A research roadmap. In *Software Engineering for Self-Adaptive Systems*, 1–26. Springer Berlin / Heidelberg.

Chihoub, H.E., Ibrahim, S., Antoniu, G., and Perez, M.S. (2012). Harmony: Towards automated self-adaptive consistency in cloud storage. In *CLUSTER*.

Clune, M., Mosterman, P., and Cassandras, C. (2006). Discrete event and hybrid system simulation with SimEvents. In *8th International Workshop on Discrete Event Systems*, 386–387.

Diao, Y., Hellerstein, J.L., Parekh, S., Griffith, R., Kaiser, G.E., and Phung, D. (2006). A control theory foundation for self-managing computing systems. *IEEE J.Sel. A. Commun.*, 23(12), 2213–2222.

Diniz, P.C. and Rinard, M.C. (1997). Dynamic feedback: an effective technique for adaptive computing. In *PLDI*.

Fielding, R.T. and Taylor, R.N. (2002). Principled design of the modern web architecture. *ACM Trans. Internet Technol.*, 2(2), 115–150.

Filieri, A., Ghezzi, C., Leva, A., and Maggio, M. (2011). Self-adaptive software meets control theory: A preliminary approach supporting reliability requirements. In *ASE*, 283–292.

Guo, Z., McDirmid, S., Yang, M., Zhuang, L., Zhang, P., Luo, Y., Bergan, T., Bodik, P., Musuvathi, M., Zhang, Z., and Zhou, L. (2013). Failure recovery: when the cure is worse than the disease. In *HotOS*, 8–14.

Hellerstein, J.L. (2010). Why feedback implementations fail: the importance of systematic testing. In *FEBID*.

Kephart, J.O. (2005). Research challenges of autonomic computing. In *ICSE*, 15–22.

Klein, C., Maggio, M., Årzén, K.E., and Hernández-Rodriguez, F. (2014). Brownout: Building more robust cloud applications. In *ICSE*. Available as preprint at: `http://goo.gl/0jMz9S`.

Kramer, J. and Magee, J. (2007). Self-managed systems: an architectural challenge. In *FOSE*, 259–268.

Misailovic, S., Sidiroglou, S., Hoffmann, H., and Rinard, M. (2010). Quality of service profiling. In *ICSE*, 25–34.

Nah, F.F.H. (2004). A study on tolerable waiting time: how long are web users willing to wait? *Behaviour and Information Technology*, 23(3), 153–163.

Patikirikorala, T., Colman, A., Han, J., and Wang, L. (2012). A systematic survey on the design of self-adaptive software systems using control engineering approaches. In *SEAMS*, 33–42.

Philippe, J., De Palma, N., Boyer, F., and Gruber, O. (2010). Self-adaptation of service level in distributed systems. *Softw. Pract. Exper.*, 40(3), 259–283.

Weyns, D., Iftikhar, M.U., de la Iglesia, D.G., and Ahmad, T. (2012). A survey of formal methods in self-adaptive systems. In *C3S2E*, 67–79.

Zhu, X., Uysal, M., Wang, Z., Singhal, S., Merchant, A., Padala, P., and Shin, K. (2009). What does control theory bring to systems research? *SIGOPS Oper. Syst. Rev.*, 43(1), 62–69.