BrownoutCC: Cascaded Control for Bounding the Response Times of Cloud Applications

Nylander, Tommi; Klein, Cristian; Årzén, Karl-Erik; Maggio, Martina

Published in:
Proceedings of the American Control Conference

DOI:
10.23919/ACC.2018.8431282

2018

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Brownout^{CC}: Cascaded Control for Bounding the Response Times of Cloud Applications

Tommi Nylander[1], Cristian Klein[2], Karl-Erik Årzén[1], Martina Maggio[1]
[1] *Department of Automatic Control, Lund University, Sweden*
[2] *Department of Computing Science, Umeå University, Sweden*

***Abstract*— Cloud computing has emerged as an inexpensive and powerful computing paradigm, to the point that now even applications with hard deadlines are executed in the cloud. It may happen, due to unexpected events, that an application becomes popular and receives a lot of attention and client requests in a short period of time. Provisioning computing capacity for such applications is quite a difficult task, because content popularity cannot be easily predicted. One of the main problems in case content has to be served with a hard deadline is to ensure that this deadline is respected, even in the presence of popularity spikes. To this end, partial computation and graceful degradation were exploited, originating the *brownout* framework. Applications would degrade the user experience in the presence of load variations, to guarantee that deadlines are met. Two different control paradigms were applied to brownout: discrete-time control of optional content percentage over a period and event-based queue management. The first one had reasonable performance providing formal guarantees about the solution. The second one was able to improve the performance and keep the response time at the setpoint better, but suffered from the drawback of not providing formally-grounded mathematical guarantees. In this work we combine the best of both worlds, providing a cascaded controller for brownout, based on a more precise model of the cloud application with respect to the original design. The Brownout^{CC} controller achieves performance comparable with the event-based version, without sacrificing formal guarantees.**

I. Introduction

Control theory is becoming important in domains where problems were previously solved using heuristic solutions, without having access to formally grounded analysis tools. One of these is the computing systems domain [12]. Computing resource allocation can easily be cast into a control problem, where a controller decides the amount of resource to allocate to different entities based on desired and measurable performance metrics [1, 17, 21, 25]. Recently, the cloud computing domain has emerged as an interesting application domain for control-theoretical principles and techniques [4, 7, 10, 15, 22].

One of the most difficult problems in cloud computing is to quickly and effectively react in the presence of flash crowds. A flash crowd is caused by a sudden increase in popularity of some content, that is then served to millions of users at the same time. The amount of resources needed to serve this increased amount of requests is unlikely to be available, unless there was a substantial over-provisioning of computing capacity before the raise in popularity.

This work was partially supported by the Wallenberg Autonomous Systems and Software Program (WASP), by the Swedish Research Council (VR) for the projects "Feedback Computing" and "Power and temperature control for large-scale computing infrastructures", by the LCCC Linnaeus Center and, by the ELLIIT Excellence Center at Lund University.

To mitigate this problem, it is common to resort to techniques like graceful degradation. A possible way of degrading the performance of a web server is to deny admission to some of the requests when it is not possible to meet the user demands [1]. Admission control means that some users would not receive any response at all, hence risking losing them to competitors, incurring long-term revenue loss. Another possibility is to assign a maximum time to each request and iteratively refine an answer until the time budget expires [9, 14]. This strategy works well for pruning search queries of spurious results, but does not easily generalize to all types of cloud applications.

A third possibility to apply the principles of graceful degradation is called *brownout* [19, 22]. When producing the response to the user requests, it is often possible to identify a part of the response that is the necessary information the user wants to see and a part of the response that would provide a better user experience and increased revenues, but is not mandatory. In the case of a travel agency website, the mandatory part of the response is the flight search, while additional optional information are car rental locations and hotel suggestions. Clearly, the application owner wants to provide the additional information, but not at the expense of losing a customer. Brownout [19] divides the response into the two mentioned parts and measures the response time to determine if the optional content is served (at an additional computation cost) or not.

The core idea behind brownout is to serve as much optional content as possible, without penalizing response times. The cloud application uses feedback from the response times to determine how much optional content can be served without sacrificing performance. The first brownout proposal used a very simple first-order model for the system [19, 22] and proposed some control strategies. Using discrete-time control, it was possible to prove properties of the closed-loop system, like stability and zero steady-state error [19]. However, the sampling period of the controller would still be a critical parameter. Decisions would be made periodically, but depending on the arrival rate of requests at the server, the control period could either be too small, leading to taking decisions based on too few response times, or too large, leading to a large lag in controller response. This could mean deciding based on the average of many response times or of none. To avoid this disparity and gain additional performance, an event-based version of the brownout principle was then devised [8], which would take a new decision at every request arrival. The event-based brownout controller [8] showed very good performance, but lacked the mathematical formalization and analysis possibility.

The contribution of this paper is three-fold: (i) We formalize the brownout control strategy in [8] into a cascaded control problem; (ii) We design the inner and outer loop controllers, proposing both a feedback and a feedforward plus feedback version. Our controller features both the performance of [8] and the formal guarantees of [19]; (iii) We evaluate our approach and compare with previous solutions using the *brownout* simulator. Besides providing formal guarantees, our controllers show fewer oscillations and maintain the measured response times closer to the target.

II. The brownout approach

This section provides some background information about the brownout model and controllers developed in [8, 19]. It also introduces some basic terminology that will then be used to explain the Brownout$^{\text{CC}}$ approach.

A brownout-aware application generates responses that are composed of two different parts: the mandatory and the optional content. In some cases, a response is produced including both parts of the content, while in other cases, to speed up the process and consume less resources, only the mandatory part is included in the response. The aim of the brownout approach is to maintain certain statistics for the user response time. In cloud computing, the focus is on maintaining tail response time – instead of average – as it was shown to better correlate with user experience [6]. For this reason, we focus our effort on the 95$^{\text{th}}$ percentile of the response time for the user requests.

Furthermore, notice that simplicity is an important feature of every control strategy for a system like this. In fact, the control computation happens on the same hardware that provides responses to the users' requests. In case the control strategy is simple and executes fast enough, more hardware power is devoted to answering requests from actual users of the web application. Due to this remark, simplicity is one of the key points in evaluating our control strategy.

This simplicity also applies to plant modeling. In contrast to physical plants, the hardware and software stack of cloud applications are so complex that it is unfeasible to devise a detailed model. Therefore, when controlling software system, one aims for as simple plant models as possible while still capturing the essential relationship between inputs and outputs. This also implies that linear models and linear design techniques often are a good choice.

A. Original control strategy

Assume that a brownout controller is periodically selecting the probability of including the optional content in a response, called the *dimmer* value. The controller period is τ_c seconds and to each controller intervention we associate a cardinal number k. We denote by $\theta(k)$ the dimmer value that the controller computes for the interval $[(k-1)\,\tau_c, k\,\tau_c]$.

The brownout approach presented in [19] assumes that the cloud application behaves according to a very simple first-order model. According to the model, the value of the 95$^{\text{th}}$ percentile of the response time τ_{95} varies depending on the dimmer value as follows

$$\tau_{95}(k) = \phi(k-1)\,\theta(k-1) + \delta\tau_{95}(k), \tag{1}$$

where $\phi(k-1)$ is a time-varying coefficient that depends on the computing platform and can be estimated and $\delta\tau_{95}(k)$ is a disturbance, interfering with the nominal system's behavior. Loop shaping is then used to synthesize a controller for the system. We denote by $e_{\tau 95}(k)$ the error between the desired 95$^{\text{th}}$ percentile of the response time $\bar{\tau}_{95}(k)$ and the actual value. Assuming that no disturbance is acting on the system, the desired closed loop system Z-transform between the setpoint $\bar{\tau}_{95}(k)$ and the actual value of $\tau_{95}(k)$ is

$$G(z) = \frac{1-p_b}{z-p_b} \tag{2}$$

where p_b, the pole of the closed loop system, is simply a parameter of the controller. The unsaturated dimmer value $\theta^*(k)$ can then be selected as

$$\theta^*(k) = \theta(k-1) + \frac{1-p_b}{\hat{\phi}(k)}\,e_{\tau 95}(k) \tag{3}$$

where $\hat{\phi}(k)$ is an estimate of $\phi(k)$ obtained with a Recursive Least Square (RLS) filter. The dimmer value θ represents the probability of carrying out the execution of the optional content, therefore it is saturated in order to be bounded in the interval $[0, 1]$.

The expression of the closed loop system in (2) allows one to prove stability (provided that the pole p_b is chosen accordingly) and zero steady-state error (the static gain is equal to 1). The proof is subject to how well the model (1) approximates the behavior of the cloud application [19].

B. Event-driven brownout

The event-based version of the brownout paradigm [8] works as follows. A periodic controller updates a threshold value $\psi_q(k)$ for the length of the queue of requests that have not yet been answered, with period τ_c.

Assume that a request r arrives at time t_r and that time t_r is included in the control interval $[k, k+1]$. Denote with $o_r \in \{0, 1\}$ the indicator of the execution of the code for the optional content – i.e., if $o_r = 1$ the optional content is computed and if $o_r = 0$ the optional content is not computed. The web server compares the amount of requests already queuing in the system $q(t)$ and the threshold set by the controller at the closest k-th control period $\psi_q(k)$ and determines if the optional content should be provided or not.

$$\begin{array}{lcl} q(t_r) \geq \psi_q(k) & \implies & o_r = 0 \\ q(t_r) < \psi_q(k) & \implies & o_r = 1 \end{array} \tag{4}$$

This algorithm has the advantage of being very easy to implement. The threshold ψ_q was in [8] set using a manually tuned PI controller with anti-windup.

However, the absence of a proper model for the queue behavior and the application behavior creates difficulties in proving properties of the closed loop system. Empirically though, the cloud application was shown to have very good performance in terms of the 95$^{\text{th}}$ percentile of the response times being close to its desired value [8].

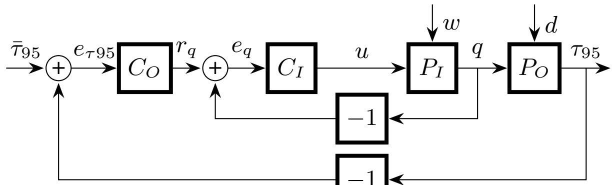


Fig. 1: A general cascaded control structure interpretation of [8].

III. The Brownout$^{\text{CC}}$ approach

This section describes the design of a brownout control strategy that combines the advantages of both the methods described in Section II, obtaining a formally analyzable controller. Subsection III-A motivates the use of a cascaded structure. Section III-B describes the inner loop, while Sections III-C, III-D, and III-E respectively discuss modeling, feedback, and feedforward control of the outer loop.

A. Event-driven brownout interpreted as cascaded control

In this section, we take a closer look at the event-driven approach in [8], that we briefly summarized in Section II-B. We here show that the threshold-based algorithm described in Equation (4) – that decides on optional content execution via the variable o_r – can be interpreted as part of a queue length control loop. In this interpretation, the threshold $\psi_q(k)$ translates to a queue length setpoint $r_q(k)$. In fact, the threshold-based approach serves optional content until the threshold $\psi_q(k)$ is reached and avoids serving optional content when the threshold is passed. The number of enqueued requests is then kept as close as possible to a function of the threshold, therefore translating it into a setpoint $r_q(k)$.

We denote by t_r the arrival time of a generic request r. At t_r, the algorithm shown in Equation (4) tries to keep the measured queue length $q(t_r)$ equal to a setpoint $r_q(t_r)$, by means of a simple on/off controller – i.e., turning on and off the computation of the optional part of the response. The controller takes as input the queue length error $e_q(t_r) = r_q(t_r) - q(t_r)$, and determines the choice of executing optional content $o_r \in \{0, 1\}$ as control signal.

This queue length control loop is driven by the request arrival, and acts at times t_r, when the request is received. To fully describe the algorithm of Section II-B, we need to complement this choice with the selection of the setpoint r_q, which as stated before, was done using a periodically executed PI controller.

The overall scheme can then be described using the cascaded structure depicted in Figure 1. In this representation, the generic control signal u (Figure 1), is the control signal o_r, C_I corresponds to the on/off controller in Equation (4) and C_O the manually tuned PI controller that selects the queue length setpoint. In [8], P_I and P_O are left unmodeled.

The cascaded interpretation in Figure 1 lays the foundation for our approach. The generic inner loop control signal u influences the response times of the cloud application, by changing the length of the queue of unserviced requests. P_I is the transfer function from the control signal determined by the controller C_I to the queue length, while P_O models the effect of the queue length on the response times. The outer loop control signal $r_q > 0$ is determined by the outer controller C_O and indicates a queue length setpoint.

To complete the model, we introduce two terms – w and d – representing disturbances acting respectively on the inner and outer loop. A web application hosted in the cloud is always subject to disturbances, such as changes in the number of users or in the computation speed. For example, additional load could be co-located with the virtual machine hosting the application, changing the efficiency of the computation resources [23]. We distinguish between two different types of disturbances: w represents a disturbance that causes the queue length q to vary due to stochastic variations (i.e. deviations from the mean) in the arrivals, d, on the contrary, is a load disturbance that causes τ_{95} to deviate even if r_q is kept constant. The control strategy, i.e., C_I and C_O, should be designed with both disturbance types in mind, in order to successfully keep τ_{95} close to its setpoint $\bar{\tau}_{95}$.

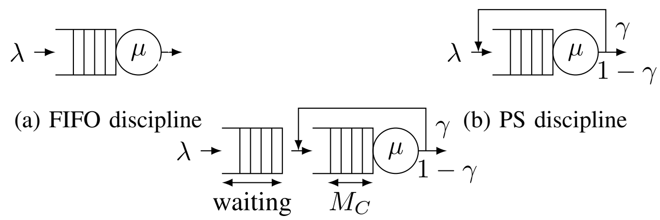
Viewing the control structure as a cascaded one has several advantages compared to single loop structure: (a) The system is faster in rejecting disturbances w acting on the inner loop; (b) The dynamics of the inner closed-loop can be linearized as shown in Section III-B; and (c) The separation of the time-scales simplifies the control design. The inner controller can be designed to reject w disturbances of a fast stochastic nature, and the outer controller can be designed to reject load disturbances d. As a drawback, the cascaded structure requires measurements of the queue lengths in addition to the response time data. However, this is easy to solve from an implementation standpoint, as all the needed variables are already used in the implementation provided with [8].

Motivated by the good performance obtained empirically with the event-based brownout controller despite the lack of modeling, and by the promising benefits of the structure, the Brownout$^{\text{CC}}$ approach uses a model-based cascaded controller design and splits the modeling of the cloud application behavior into the two introduced loops.

B. Inner loop modeling and control

In cloud computing, usually applications are modeled using principles from queuing theory [20]. We summarize in the following the background notions that inspired us in the design of the model and controller for the inner loop.

Queuing discipline models such as first-in-first-out (FIFO) and processor sharing (PS) are commonly used to model the behavior of web servers, see for example [5, 11–13]. With the FIFO model, each request is executed individually based on the order of arrival, as represented by Figure 2a. In the PS model, all the active requests are assumed to be executed simultaneously, using fractions of the computing capacity of the web server. The PS discipline can be seen as a queue where each request is processed for an (infinitely) short time-slice, and returned to the back of the queue, unless completed. From the modeling perspective, a queue that uses the PS discipline is normally seen as a queue with feedback where the single parameter, γ, represents the proportion of requests returned to the queue, as shown in Figure 2b. A third option is the use of an approach that integrates both disciplines, the Combined FIFO and Processor Sharing (CFPS) model [18]. Here, the PS queue can only hold a limited $M_C > 0$ jobs. M_C models the number of available computing entities in the computing infrastructure – number



(a) FIFO discipline (b) PS discipline

(c) CFPS discipline

Fig. 2: Queuing discipline models.

of cores, number of threads – that can be executed in parallel. Requests exceeding M_C wait in a FIFO queue. This situation is shown in Figure 2c. The CFPS model is a generalization of both FIFO and PS. These two disciplines are easily interpreted as special cases of CFPS, respectively with $M_C = 1$ and $M_C = \infty$.

For our approach to be as general as possible, we consider our application to behave as a queue with the CFPS discipline as the underlying model, without any restrictions on the value of M_C. We also avoid considering special arrival processes $A(t)$ or service time distributions $B(x)$, i.e., a G/G/1 queue.

To design a *proper* control strategy for the cascaded controller, we need a valid model for P_I in the form of a transfer function, that represents the behavior of the application queue length as a response to the control signal – $u = \theta$ in the case of the original controller [19] and $u = o_r$ for the event-based version [8]. Writing such a (linear) model using queuing principles is difficult.

Here we use queuing theory as an inspiration to select a meaningful continuous-time control signal u that would allow us to model the inner loop plant P_I using a transfer function. We define $u = v = \mathrm{d}q/\mathrm{d}t$, representing the growth rate of the queue. Using this control signal, the transfer function $P_I(s)$ from v to q becomes a simple integrator:

$$P_I(s) = \frac{1}{s}. \tag{5}$$

By utilizing the concept of feedback linearization [16], i.e., determining the choice of v and designing $C_I(s)$, we are able to linearize the inner loop and choose its dynamics. The dynamics of the closed inner loop $G_I(s)$ will affect the outer loop, leading to a desire for simplicity. To achieve this simple dynamics for $G_I(s)$, $C_I(s)$ is then chosen as a P controller with gain K:

$$C_I(s) = K. \tag{6}$$

As the process $P_I(s)$ is integrating, this simple controller is able to follow reference step changes in r_q without any stationary errors. However, these might still occur due to disturbances w entering the inner loop. The inner closed loop $G_I(s)$ becomes:

$$G_I(s) = \frac{K}{s + K}, \tag{7}$$

where the design parameter K determines the speed of the system. The complete inner loop model is shown in Figure 3.

We have now defined how to compute the control signal v. In order to complete the inner loop control, we should also specify how to *actuate* it. Our controller is realized using a periodic sampling strategy, with the actuation relying on the threshold-based algorithm (4). For each sampling period h:

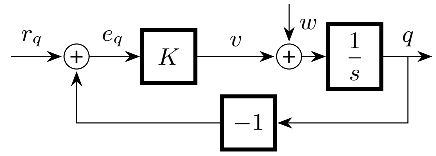


Fig. 3: Inner loop control using feedback linearization.

(i) At the beginning of the sampling period h, i.e., at time t_a, the controller (6) calculates a control signal $v(t_a)$. The control signal represents the derivative of the queue length that we desire to actuate;
(ii) A queue length threshold $\psi_q(t_a)$ is set as: $\psi_q(t_a) = q(t_a) + v(t_a)$;
(iii) For all incoming requests during h, the algorithm in Equation (4) is used, for each request, to determine if optional content should be served or not;
(iv) This strategy ensures[1] that the new queue length $q(t_a + h)$ stays close to $q(t_a) + v(t_a)$, actuating $v(t_a)$.

On the negative side, the actuation strategy is not exact, i.e., it does not guarantee to exactly actuate v, as, e.g., the arrivals $A(t)$ enter the queue according to some general random process. These deviations from the intended queue growth rate caused by actuation errors can be seen as part of the disturbance w, entering as shown in Figure 3. On the positive side, the algorithm above actuates the control signal v well, regardless of both M_C, arrival process, and service time distribution. It also reacts quickly to stochastic changes in the system, like modifications of the arrival rate – thanks to its event-driven execution. Finally, it is also very simple to implement and requires minimal execution time.

After testing the inner controller in simulations, using different values of M_C, we choose $K = 1$ as the best fit for the inner loop design. The closed inner loop then becomes:

$$G_I(s) = \frac{K}{s + K} = \frac{1}{s + 1}. \tag{8}$$

C. Outer loop modeling

To describe the outer open loop $G_P(s)$, i.e. from r_q to the response times, we split the model into two parts: (i) from r_q to q and; (ii) from q to the response times. The first part is completely described by the inner closed loop $G_I(s)$.

To model the second part we need to define precisely the meaning of "response times". We denote by τ_{95}^m the 95^{th} percentile of the response times served only with mandatory content and by τ_{95}^o the 95^{th} percentile of the response times of the requests served with mandatory and optional content. The mandatory τ_{95}^m and optional τ_{95}^o response times are expected to diverge depending on the value of M_C. The larger M_C becomes, the more the requests spend time being processed in the PS queue rather than waiting in the FIFO queue. As the mean service times are assumed to be related

[1] Assume that the mean inter-arrival times are denoted by $\bar{t}$, the mean mandatory and optional content service times respectively by $\bar{x}_m$ and $\bar{x}_o$, and that $\bar{x}_m < \bar{t} < \bar{x}_o$ holds. If the last assumption does not hold, brownout cannot find a feasible solution, and the inter-arrival times have to be adjusted to fit this assumption by e.g. adding or removing servers. According to Equation (4), mandatory content $o_r = 0$ is chosen for all $q(t_r) > \tau_q(k)$. Then, the queue length is "stable", i.e., kept close to (or slightly above) the threshold $\tau_q(k)$, since $\bar{x}_m < \bar{t}$. For a proof, see [20]. Since optional content $o_r = 1$ is chosen for all $q(t_r) \leq \tau_q(k)$, the queue stays within a bound, ξ, around $\tau_q(k)$ since $\bar{x}_o > \bar{t}$ below the threshold.

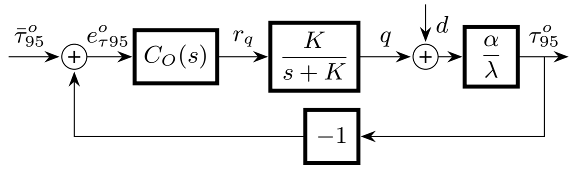


Fig. 4: The outer loop model.

as $\bar{x}_m \ll \bar{x}_o$, a high value of M_C causes the mandatory τ_{95}^m and optional τ_{95}^o response times to diverge. Since we can only act on the optional response times, we measure and use for feedback only the optional response times τ_{95}^o.

Then, the model of the second part, i.e., from q to τ_{95}^o, corresponding to the P_O block in Figure 1, can be inspired by Little's Law $\bar{\tau} = \bar{q}/\lambda$ [20]. Here $\bar{\tau}$ and $\bar{q}$ represent mean response times and queue lengths, and λ represents the mean arrival rate. Instead of mean values, we want to model the 95$^{\text{th}}$ percentile of the response times. The theorem is thus not directly applicable, but it serves as a good approximation when we introduce a correction term, that we denote by α. The following static relation from q to τ_{95}^o is then proposed:

$$P_O(s) = \frac{\alpha}{\lambda}. \tag{9}$$

Here the constant α is assumed to vary with λ and M_C. The complete outer loop model is shown in Figure 4.

D. Design of outer loop feedback controller

The task is to design the outer loop controller $C_O(s)$, given the open loop transfer function from r_q to τ_{95}^o as

$$G_P(s) = \frac{K}{s+K}\frac{\alpha}{\lambda} = \frac{1}{s+1}\frac{\alpha}{\lambda}, \tag{10}$$

using $K = 1$ as chosen in Section III-B. We design the controller using pole-placement. As $G_P(s)$ is a first order system, the poles can be placed arbitrarily using only two controller parameters. In addition, the controller should be able to reject load disturbances d, resulting from stationary errors in the inner loop as well as from changes in the load. Furthermore, the controller should be able to handle the fact that α is unknown and varying and cope with changes in the process gain $G_P(0)$, especially since the arrival rate λ is expected to vary over time. The proposed solution is to select the controller parameters assuming a nominal process gain G_N. The adaptive controller gain k_a is then adjusted in order to counteract multiplicative changes to $G_P(0)$, such that $G_P(0)\, k_a \approx G_N$, giving the adaptive PI controller:

$$C_O(s) = k_a \left(k_p + \frac{k_i}{s} \right). \tag{11}$$

Here, $k_a = G_N/\hat{G}_P(0)$, where $\hat{G}_P(0)$ is estimated as described in Section IV. As $G_P(0)$ might change rapidly, it is not certain that k_a is able to adapt accordingly. Also, other model uncertainties might occur, requiring a robust design. For the nominal design, $\alpha = 1$ and $\lambda = 20$ are chosen giving $G_N = 0.05$, and the nominal process $G_P^N(s)$ as

$$G_P^N(s) = \frac{0.05}{s+1}. \tag{12}$$

The poles of the outer closed loop system are placed according to the characteristic equation

$$s^2 + 2\zeta\omega_O s + \omega_O^2, \tag{13}$$

where $0 \leq \zeta \leq 1$ is the relative damping and ω_O the speed of the outer loop. In order to ensure a robust design, $\zeta = 1$ is chosen placing the poles on the negative real axis as $(s+\omega_O)^2$. The choice of ω_O results in a trade off between robustness and noise rejection as the maximum M_S of the sensitivity function S in this case decreases when ω_O grows. As a result, $\omega_O = 0.6$ is chosen, setting the speed of the outer loop to about half the speed of the inner loop ($\omega_I = 1$). The choice also ensures good robustness properties as $M_S = 1.03$. This results in the controller parameters $k_p = 4.0$ and $k_i = 7.2$, the adaptive PI controller equation becoming

$$C_O(s) = \frac{0.05}{\hat{G_P}(0)} \left(4.0 + \frac{7.2}{s} \right). \tag{14}$$

The derived controller is fairly standard. However, in our opinion this is only an advantage made possible by the cascaded structure. Using such a simple controller allows us not to waste computational power, that the application could use to serve user requests.

E. Design of outer loop feedforward controller

Testing the feedback controller of Section III-D, we have experienced the need for a better disturbance rejection mechanism for the outer loop. We achieve this with the design of a standard feedforward controller.

Equation (10) shows the outer open loop process dynamics $G_P(s)$. Selecting $\tau_{95}^o = \bar{\tau}_{95}^o$, as well as considering the dynamics in (10) in stationarity, leads to a proposed static feedforward scheme

$$r_q^{ff} = \frac{1}{\hat{G_I}(0)} \frac{\hat{\lambda}}{\hat{\alpha}} \bar{\tau}_{95}^o. \tag{15}$$

Here $\hat{G_I}(0)$, $\hat{\lambda}$ and $\hat{\alpha}$ are estimated as described in Section IV. The feedforward scheme (15) is combined with the feedback controller designed in the previous section, resulting in the complete control structure shown in Figure 5.

IV. Evaluation

This section presents our results. We validate our control strategy using the open source Python-based brownout simulator[2], built to mimic the behavior of cloud applications [10] and described in the following Section IV-A.

A. The simulator

The simulator defines the concepts of *Client*, *Request*, *Replica* – a single server, running a brownout application – and *Replica Controller*. Clients issue requests to be served by the replica (server). Clients can behave according to the open-loop or to the closed-loop client model [2, 24]. In the closed-loop model, clients wait for a response and issue a new request only after some think time. In the open loop model, clients do not wait and instead issue new requests with a specific request rate. Being better at modelling a large number of independent users, we performed the evaluation with open-loop clients.

For each request, the simulator computes the service time. The time it takes to serve requests with only the mandatory or with the optional content in addition to the mandatory one are computed as random variables, with normal distributions, whose mean and variance are based on profiling data from

[2]https://github.com/cloud-control/brownout-lb-simulator

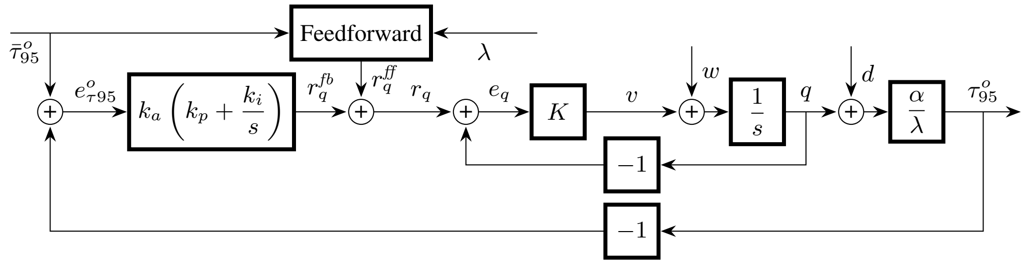


Fig. 5: The complete cascaded structure, with the proposed models and controllers for both outer and inner loop.

the execution of experiments on a real machine [19]. The processing time for a request with optional content is a random variable $Y \sim \mathcal{N}(0.07, 0.01)$, while the processing time for the mandatory content is a random variable $Z \sim \mathcal{N}(0.001, 0.001)$. Furthermore, the simulator supports the CFPS queuing discipline with any M_C.

Finally, replicas implement a replica controller, that takes care of selecting – for each request – when to serve optional content. In the simulator, we implemented our own replica controller, described in Section III. The controller code developed in the simulator can be directly plugged into brownout-aware applications like RUBiS[3] and RUBBoS[4]. For the controller implementation, the adaptive PI controller in (14) was discretized with sample period $h = 0.5$ s using the method suggested in [3], and complemented by a tracking-based anti-windup solution. The parameter estimations that the feedback and feedforward schemes require ($\hat{G}_P(0)$, $\hat{G}_I(0)$, $\hat{\lambda}$, $\hat{\alpha}$) are implemented as exponentially weighted moving averages according to

$$\hat{y}(k+1) = \beta \hat{y}(k) + (1-\beta)\, y(k). \tag{16}$$

Here $\hat{y}_k$ is the estimate of y, y_k the measurement at time k and $0 \le \beta \le 1$ a design parameter. In our simulations we use slightly different β values for the different parameters that we estimate, but mostly $\beta \simeq 0.9$.

B. Control validation

The response time requirements of the application are expressed in the form of a maximum value for the 95^{th} percentile of the response times. To bound this value, the controller should be able to constrain the 95^{th} percentile of the response times for the requests that are served with optional content, τ^o_{95}. The remainder of this evaluation focuses on τ^o_{95}, and uses a setpoint $\bar{\tau}^o_{95} = 1$ s.

The adaptive PI controller (denoted by C_{fb}) derived in Section III-D is compared with the combined feedback+feedforward scheme (denoted by C_{ff}) from Section III-E, as well as with the original brownout design (denoted by C_{orig}) described in Section II-A and the event-based design[5] (denoted by C_{event} and described in Section II-B). Since no clear tuning rules were proposed in [8], we have tuned its outer controller in the same way as C_{fb} without the adaptive gain. As anticipated, the simulations are performed with Poisson arrivals generated by open-loop clients, and with both $M_C = 3$ and $M_C = 10$, respectively representing behaviors close to FIFO and PS.

Figures 6 to 11 show a simulated sequence (repeated 20 times for statistical significance) of varying arrival rates for both values of M_C. The arrival rates vary following the sequence $\{20, 100, 30, 70, 20\}$ s^{-1}, representing step changes in the load d, and each value is kept constant for 60 s. Figures 6 to 9 show the 95% confidence intervals of the plotted quantities. The upper plots show the derivative of the queue length (i.e., the $v = \dot{q}$ control signal), displaying both the computed (v) and the actuated control signal (v_{actual}). The middle plots show the actual queue length q and its reference value r_q (i.e., the outer loop control signal). Finally, the bottom plots display the response times τ^o_{95} and its setpoint $\bar{\tau}^o_{95} = 1$. Figures 10 and 11 show a comparison of all the four strategies. The upper plots represent the dimmer value θ (i.e., the percentage of requests served with optional content), which is determined by the controller in the case of C_{orig} and *a posteriori* computed in the case of the other strategies. The middle plots show the reference values of the queue length r_q for the proposed strategies (C_{fb}, C_{ff}) and for the event-based controller (C_{event}). The lower plots show τ^o_{95} and its setpoint $\bar{\tau}^o_{95} = 1$. The plots of Figures 10 and 11 show average values over the 20 repeated sequences for readability.

Figures 6 to 9 show one of the benefits of a cascaded structure: the inner loop can be very fast[6], allowing a tighter control. In some cases (e.g., Figures 7–9, in the time interval 60 s–120 s) the system experiences some actuation errors, leading to a stationary error in the inner loop. Thanks to the integral action in the outer controller, response times τ^o_{95} are still kept close to their setpoints. In fact, the inner loop is in general able to follow the outer loop control signal r_q and drive the queue length q to acceptable values. The good control performance that we experience can be linked directly to the model being a better approximation compared to previous models [19]. The cascaded structure C_{fb} shows no overshoot in the queue setpoints but is slower in responding to changes in d, while C_{ff} is faster in handling changes in the arrival rates, but overshoots. For $M_C = 10$, the C_{ff} controller gets larger overshoots in its outer loop control signal r_q, as a result of the assumed model (10) not describing the dynamics as well as for $M_C = 3$. However, this has a minimal effort on the control performance.

Looking at Figures 10 and 11, the amount of optional content served (shown in the θ plot) is an indication of how

[3]https://github.com/cloud-control/brownout-rubis
[4]https://github.com/cloud-control/brownout-rubbos
[5]In the event-based design, we use τ^o_{95} as measurement signal, for fairness with respect to our solution.
[6]As there is no physical actuator involved, the aggressive behavior is not an issue.

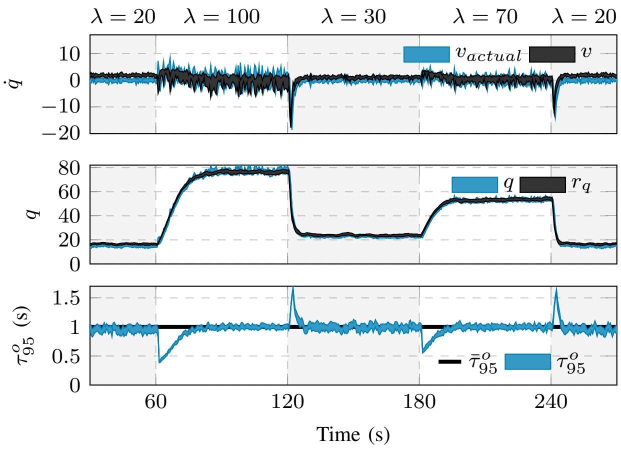


Fig. 6: 95% confidence interval plots for C_{fb} with $M_C = 3$.

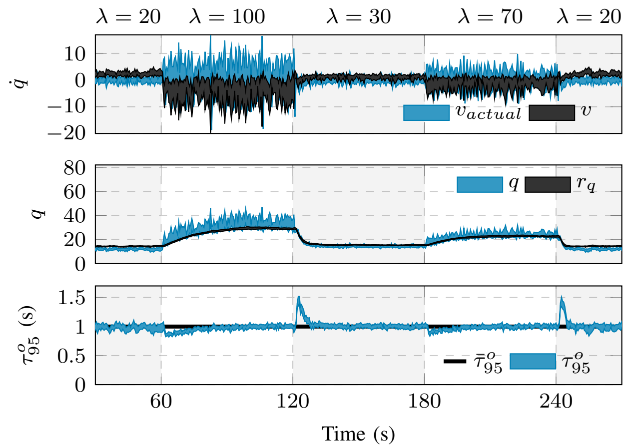


Fig. 7: 95% confidence interval plots for C_{fb} with $M_C = 10$.

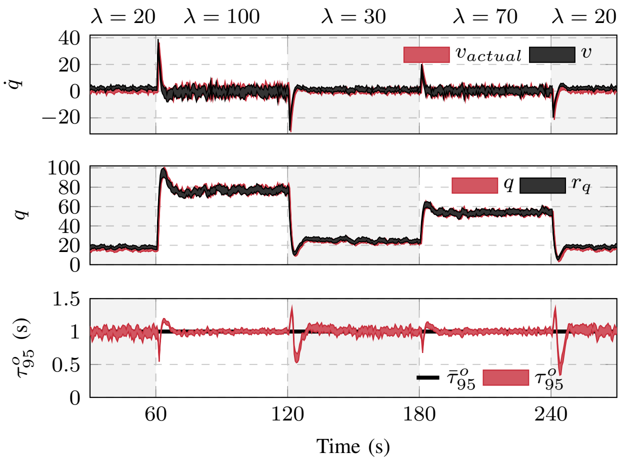


Fig. 8: 95% confidence interval plots for C_{ff} with $M_C = 3$.

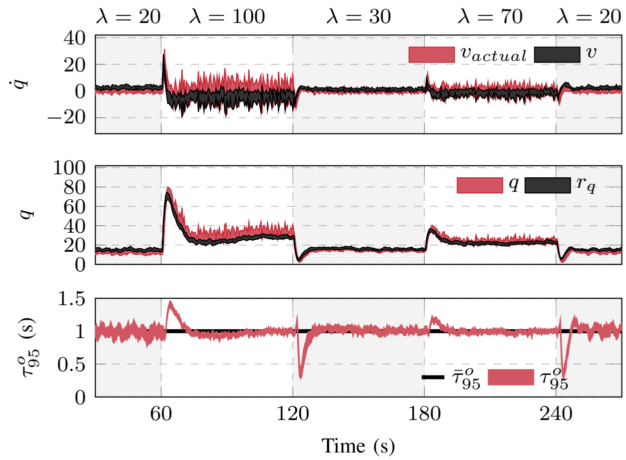


Fig. 9: 95% confidence interval plots for C_{ff} with $M_C = 10$.

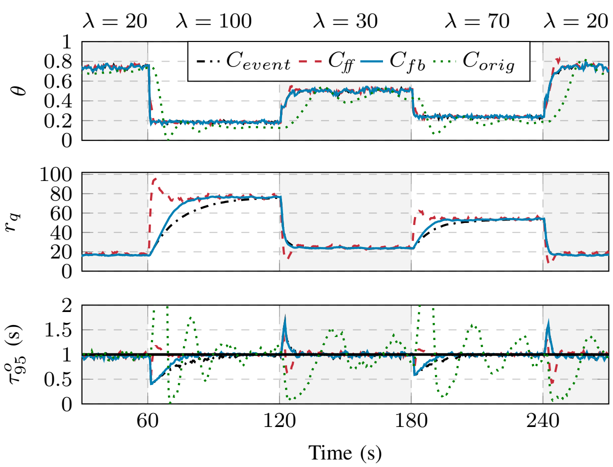


Fig. 10: Average value plots for all 4 strategies with $M_C = 3$.

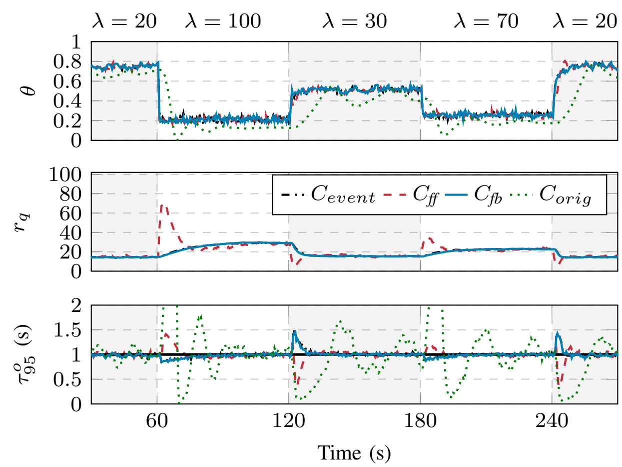


Fig. 11: Average value plots for all 4 strategies with $M_C = 10$.

well the application behaves in terms of potential revenues for the application owner. Both in the case of the event-based strategy C_{event} and our proposals C_{fb} and C_{ff}, the average dimmer value is 28%, while the original strategy C_{orig} only achieves an average value of 25% optional content served. Quite naturally, as the arrival rate increases the amount of optional content served decreases. As a result of the robust design, the control performances of C_{fb} and C_{ff} are able to serve additional optional content, while keeping the response times around the setpoint under the different conditions, clearly outperforming the original design. Note that the results are truncated for C_{orig}, its peak values of τ^o_{95} reaches about 5 seconds for both M_C.

Table I presents quantitative data comparing the four strategies in the same 20 repeated simulations, for both values of M_C. The first two columns show the Integral of the Absolute Error ($\int |e^o_{\tau 95}(t)| dt$), the following columns show the variance of *all* the optional content response times τ^o

TABLE I: Quantitative comparison of all 4 strategies.

M_C	IAE $[\cdot 10^3 \mathrm{s}]$ 3	IAE 10	var(τ^o) [s] 3	var(τ^o) 10	τ^o_{max} [s] 3	τ^o_{max} 10
C_{orig}	8.23	8.34	0.695	0.745	5.68	6.27
C_{event}	1.58	1.01	0.031	0.021	1.82	1.93
C_{fb}	1.48	0.98	0.030	0.021	1.81	1.83
C_{ff}	1.23	1.43	0.026	0.034	1.56	1.65

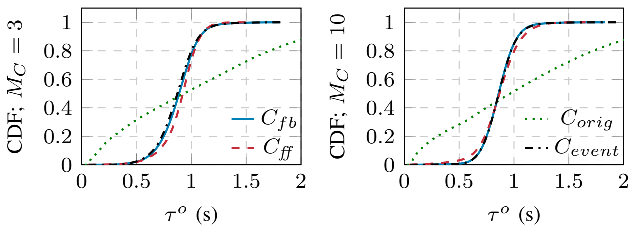


Fig. 12: Empirical cumulative distributions of τ^o for all 4 strategies.

and the last two columns show the maximum value of τ^o. The developed controllers are very close to the event-based controller [8], but provide formal guarantees. Also, especially with C_{ff} the maximum response time is lower than with C_{event}.

Finally, to complete our evaluation, we computed the empirical Cumulative Distribution Function (CDF) of τ^o for both values of M_C, using the four control strategies. Also in this case, the simulations are repeated 20 times (but not averaged). Figure 12 shows the results, indicating clearly that the control strategies synthesized in this paper outperform the original design [19] C_{orig}, and behave similarly to the event based controller C_{event}. C_{ff} and C_{fb} display much shorter tails in the response times, and are able to keep the 95$^{\text{th}}$ percentile close to 1 second. Finally, C_{ff} is able to keep the tails slightly shorter than C_{fb}, thanks to its faster reactions to changes in the arrival rate.

V. Conclusion and Future Work

In this paper a novel brownout controller was presented, capable of combining the benefits of both the event-based brownout [8] in terms of performance and the advantages of the original approach [19], in terms of analysis.

This research was motivated by the desire of solving the autoscaling problem for brownout applications – i.e., to decide when to start a new virtual machine for the same cloud application, taking also advantage of the knowledge of the dimmer value and not only of the response times. We have realized that the brownout loop, in any of its forms, was not suitable for being directly extended with autoscaling capabilities and there was a need for a more realistic model of the behavior of the application. Together with a better control strategy, this paper provides such a model, which we plan to use for brownout-aware autoscaling.

References

[1] T. F. Abdelzaher et al. “Performance guarantees for Web server end-systems: a control-theoretical approach”. In: *IEEE Transactions on Parallel and Distributed Systems* 13.1 (Jan. 2002).

[2] F. Alomari and D. A. Menasce. “Efficient Response Time Approximations for Multiclass Fork and Join Queues in Open and Closed Queuing Networks”. In: *IEEE Trans. Parallel Distrib. Syst.* 25.6 (June 2014).

[3] K. J. Åström and B. Wittenmark. *Computer-controlled Systems (3rd Ed.)* Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1997.

[4] C. Barna et al. “Cloud Adaptation with Control Theory in Industrial Clouds”. In: *2016 IEEE International Conference on Cloud Engineering Workshop, IC2E Workshops, Berlin, Germany, April 4-8, 2016*. 2016.

[5] J. Cao et al. “Web server performance modeling using an M/G/1/K* PS queue”. In: *Telecommunications, 2003. ICT 2003. 10th International Conference on*. Vol. 2. IEEE. 2003.

[6] J. Dean and L. A. Barroso. “The Tail at Scale”. In: *Commun. ACM* 56.2 (Feb. 2013).

[7] K. M. Deliparaschos et al. “On the use of fuzzy logic controllers to comply with virtualized application demands in the cloud”. In: *2016 European Control Conference, ECC 2016, Aalborg, Denmark, June 29 - July 1, 2016*. 2016.

[8] D. Desmeurs et al. “Event-Driven Application Brownout: Reconciling High Utilization and Low Tail Response Times”. In: *2015 International Conference on Cloud and Autonomic Computing*. 2015.

[9] S. Ding et al. “Indexing Strategies for Graceful Degradation of Search Quality”. In: *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*. SIGIR ’11. Beijing, China: ACM, 2011.

[10] J. Durango et al. “Control-theoretical load-balancing for cloud applications with brownout”. In: *53rd IEEE Conference on Decision and Control, CDC 2014, Los Angeles, CA, USA, December 15-17, 2014*. 2014.

[11] V. Gupta et al. “Analysis of join-the-shortest-queue routing for web server farms”. In: *Performance Evaluation* 64.9 (2007).

[12] J. L. Hellerstein et al. *Feedback Control of Computing Systems*. John Wiley & Sons, 2004.

[13] L. Hoflack et al. “Modeling Web Server Traffic with Session-Based Arrival Streams”. In: *Proceedings of the 15th International Conference on Analytical and Stochastic Modeling Techniques and Applications*. ASMTA ’08. Nicosia, Cyprus: Springer-Verlag, 2008.

[14] V. Jalaparti et al. “Speeding Up Distributed Request-response Workflows”. In: *Proceedings of the ACM SIGCOMM 2013 Conference on SIGCOMM*. SIGCOMM ’13. Hong Kong, China: ACM, 2013.

[15] E. Kalyvianaki et al. “Adaptive Resource Provisioning for Virtualized Servers Using Kalman Filters”. In: *TAAS* 9.2 (2014).

[16] H. K. Khalil. “Noninear Systems”. In: *Prentice-Hall, New Jersey* 2.5 (1996).

[17] M. A. Kjaer et al. “Resource allocation and disturbance rejection in web servers using SLAs and virtualized servers”. In: *IEEE Transactions on Network and Service Management* 6.4 (Dec. 2009).

[18] M. A. Kjaer et al. “Response-time control of a single server queue”. In: *2007 46th IEEE Conference on Decision and Control*. Dec. 2007.

[19] C. Klein et al. “Brownout: Building More Robust Cloud Applications”. In: *Proceedings of the 36th International Conference on Software Engineering*. ICSE 2014. Hyderabad, India: ACM, 2014.

[20] L. Kleinrock. *Queueing Systems*. Vol. I: Theory. Wiley Interscience, 1975.

[21] C. Lu et al. “Feedback Control Real-Time Scheduling: Framework, Modeling, and Algorithms*”. In: *Real-Time Systems* 23.1 (2002).

[22] M. Maggio et al. “Control strategies for predictable brownouts in cloud computing”. In: *IFAC Proceedings Volumes* 47.3 (2014). 19th IFAC World Congress.

[23] J. Mars et al. “Bubble-Up: Increasing Utilization in Modern Warehouse Scale Computers via Sensible Co-locations”. In: *Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture*. MICRO-44. Porto Alegre, Brazil: ACM, 2011.

[24] B. Schroeder et al. “Open Versus Closed: A Cautionary Tale”. In: *Proceedings of the 3rd Conference on Networked Systems Design & Implementation - Volume 3*. NSDI’06. San Jose, CA: USENIX Association, 2006.

[25] S.-Y. Yun and A. Proutiere. “Distributed Proportional Fair Load Balancing in Heterogenous Systems”. In: *Proceedings of the 2015 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems*. SIGMETRICS ’15. Portland, Oregon, USA: ACM, 2015.