Adaptive Run-time Performance Optimization Through Scalable Client Request Rate Control

Guenther Starnberger, Lorenz Froihofer, and Karl M. Goeschka
Vienna University of Technology
Information Systems Institute, Distributed Systems Group
Argentinierstrasse 8/184-1
1040 Vienna, Austria
{guenther.starnberger, lorenz.froihofer, karl.goeschka}@tuwien.ac.at

ABSTRACT

Today's Internet-scale computing systems often run at a low average load with only occasional peak performance demands. Consequently, computing resources are often overdimensioned, leading to high costs. While load control techniques between clients and servers can help to better utilize a given system, these techniques can place a significant communication and computation load on servers. To improve on these issues, we contribute with scalable techniques for client-request rate control, achieved through integration of (i) a scalable distributed feedback channel to transmit control information from the server to the clients with (ii) decoupling strategies that allow to constrain and filter client requests directly at the client side, illustrated in the area of first-price sealed-bid online auctions, and (iii) a PID controller that adaptively controls the input parameters of those decoupling strategies to facilitate an optimal server utilization. In contrast to related work, we can hence optimize server load directly at the source through rate control of the clients. Our evaluations show that this setup supports large sets of clients before the controller becomes unstable.

Categories and Subject Descriptors

C.2.4 [**Distributed Systems**]: Client/Server

General Terms

Performance, Measurement, Reliability

Keywords

Performance optimization, load control, temporal decoupling

1. INTRODUCTION

Performance and availability are two core concerns in today's Internet applications. In this paper we address these points for certain types of Internet applications. Our primary application scenario focuses on first-price sealed-bid auctions: All bids submitted before a certain deadline are accepted, while all bids submitted after this deadline are rejected. As users do not learn about each others bids before the deadline, there is no direct interaction between users. This allows us to decouple the auction deadline from bid submission, because clients can locally timestamp bids using a smart card and a secure time synchronization protocol [23] and transmit those bids to the server at a later time. In case of server overload or server outages, clients can therefore defer the transmission of locally timestamped bids, thus *decoupling* bid submission from bid transmission. Such first-price sealed bid auctions differ from eBay-style auctions, where our temporal decoupling approach is not applicable. In addition, multiple auctions do not overlap, but are executed sequentially. Our industrial partners estimate a global market size of at least 4.05 billion Euros for first-price sealed-bid auctions, mainly due to state bonds.

With our decoupling approach we increase performance, scalability, and availability of the system by mitigating peak loads during an auction's deadline that would otherwise require additional hardware resources. As an example of the relevance of this problem, consider the talk of Google engineer Luiz André Barroso at the O'Reilly Velocity 2008 conference[1] who stated that Google's servers are dormant most of the time with only occasional spikes of peak activity. Our decoupling technique approaches the problem by distributing requests in the temporal instead of the spatial domain. Thereby, we can avoid peak loads and facilitate more efficient use of the system's capacity, by adding the temporal dimension to the trade-off between performance and energy efficiency.

To specify *when* a client is allowed to transmit information, we introduce two decoupling strategies that defer requests directly at the client. The first decoupling strategy specifies which clients are allowed to transmit information at a particular point in time, while the second decoupling strategy controls the transmission rate of clients. For both decoupling strategies the input parameters can be adapted to alter the load at the server. To prevent overload and to enable an optimal utilization of the server we combine the decoupling mechanisms with a server-side PID (Proportional-Integral-Derivative) controller. To minimize the additional load at the server caused due to operation of the controller's feedback channel we use a *distributed* feedback channel operated mainly by the clients themselves. Hence, the server-side cost

[1]see `http://en.oreilly.com/velocity2008/public/schedule/detail/3694`

for operation of the feedback channel becomes independent from the total number of clients, allowing for better performance during peak loads than state-of-the-art systems.

To summarize, our main contribution is the integration of (i) a distributed feedback channel to transmit control information from the server to the clients with (ii) decoupling strategies that allow to constrain client requests directly at the client side and (iii) a PID controller that adaptively controls the input parameters of those decoupling strategies to facilitate an optimal server utilization.

Section 2 defines the application scenario in more detail, followed by Section 3 presenting our decoupling strategies. Section 4 continues with our request rate control loop and the distributed feedback channel, and Section 5 evaluates the effectiveness of our approach, complemented by Section 6 discussing the limitations. We discuss related work in Section 7 and conclude the paper in Section 8.

2. APPLICATION SCENARIO

In this section we first discuss first-price sealed-bid auctions which are our main application scenario. We then proceed with an examination of temporal decoupling, which allows to reduce peak-loads by spreading requests in the temporal domain. We conclude the section with a discussion about client-side security.

While we discuss temporal decoupling in the context of online auctions, this approach is also applicable to other scenarios, such as online lotteries and online gaming [15], where data transmission can be delayed, if exact and secure client-side timestamps are available.

2.1 First-price sealed-bid auctions

In first-price sealed-bid auctions the time of bid placement does not have an influence on the auction outcome, as long as bids are placed before the fixed auction's deadline, and bidders do not learn about each other's bids until the end of the auction. Furthermore, bidders are allowed to submit updates to their bids until the auction deadline. Examples of such first-price sealed bid auctions are governmental bond auctions and CO_2 certificate auctions. The characteristics of such first-price sealed-bid auctions differ from eBay-style auctions, where our temporal decoupling approach is not applicable. However, our industrial partners estimate a global market size of at least 4.05 billion Euros for first-price sealed-bid auctions, mainly due to state bonds.

In this paper we consider first-price sealed-bid auctions as used for the auctioning of bonds. These auctions exhibit high peak loads around the auction deadline, as a majority of bidders tries to submit bids shortly before the deadline. Moreover, these auctions also exhibit high dependability requirements as an auction canceled due to technical reasons can lead to significant financial losses, because market conditions change over time. Unlike in the case of eBay-style auctions, only few non-overlapping auctions are conducted per year. Therefore, it is not possible to schedule multiple auctions in a way that the combination of the individual auction peaks produces a more or less constant load, which would lead to good overall server utilization.

As a consequence, systems need to be designed for excessive peak loads and high availability, leading to massive over-provisioning and excessive costs. While cloud services would allow to mitigate some of the availability and performance issues, the high trust requirements and the high monetary amounts eliminate cloud services as an option for deployment.

2.2 Temporal decoupling

In previous work [20] we have outlined our basic idea to temporally decouple bid submission at the client from bid transmission to the server. In this paper, our first contribution elaborates on this idea and presents different strategies to actually implement the basic idea (see Section 3). In this approach we improve dependability and resource utilization, but at the same time increase the demands for security (see Section 2.3). To facilitate this approach, we apply secure client-side timestamps [23] that can then be used by the server to verify the time of bid submission.

An example of temporal decoupling is given in Figure 1. The grey line depicts the amount of bids placed by the users in a first-price sealed-bid auction, while the black line indicates the amount of bids transmitted to the server. The deadlines for both lines are different. Thus, once the user places a bid locally, it can be transferred to the server at a later point in time. Due to the longer duration for bid transmission, the peak load can be decreased, as the bids can be spread in the temporal domain. After receiving the bids, the server can read the secure timestamps assigned by the user's smart card, to ensure that the bid has been placed before the deadline for bids. However, bid transmission cannot be delayed indefinitely, as the server cannot determine the winner of an auction without knowledge of the individual bids. Thus, there is a second deadline for messages that constitutes the point in time where the temporally decoupled messages need to be received at the server.

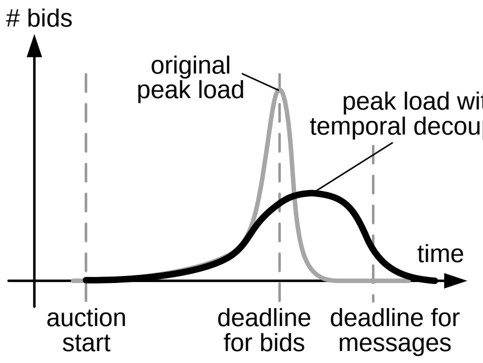


Figure 1: Effect of temporal decoupling. The *deadline for bids* specifies the deadline until which a user can place a bid, while the *deadline for messages* specifies the deadline until which the bid has to be received at the server.

By decoupling the deadline for bids from the deadline for message reception at the server (as depicted in Figure 1), we increase system performance, scalability, and availability during peak loads:

1. We increase performance and scalability, as we can intentionally delay transmission of bids during brief periods of peak loads. This decreases peak load at the server and and increases the amount of bids that can be placed at clients.

2. We increase availability, as in case of problems we can

allow bids to be transferred to the server at a later time.

For the availability, it is sufficient to apply correct timestamps locally. If the system works, we can immediately transfer bids to the server. For performance and scalability we additionally use a decoupling strategy in combination with a PID controller and a distributed feedback channel, which enables an optimal utilization of the server.

While our approach increases the effective performance of the system under peak loads, it also introduces new security challenges, as malicious bidders could tamper with the timestamp, thereby cheating the system. Therefore, we do not define a global deadline for messages—as in the simplified example given above—but instead define an individual deadline for each single message by restricting the maximum transmission delay of each message. An example of such a maximum transmission delay is the delay given by one of the decoupling strategies discussed in Section 3 plus a fixed amount of time sufficient for transmission to the server. Thus, an attacker trying to manipulate timestamps has less time to tamper with bid information stored on the smart card and cannot assign arbitrary values to timestamps. In case of server failures that prevent clients from abiding to these deadlines, the individual deadlines can be adaptively prolonged on the server-side by the auctioneer.

2.3 Client-side security

In our system, security critical components are executed within secure smart cards, including software to maintain the current time with a secure time synchronization protocol [23]. The smart card is also responsible for the application of tamper-resistant and accurate timestamps to individual bids. Therefore, our system does not depend on cooperative users, as only the smart card is able to assign timestamps accepted by the server.

The code responsible for temporal decoupling is not running within a smart card, but directly running on each user's client. Therefore, a user could theoretically manipulate the software and place bids without abiding to the decoupling strategies introduced in Section 3. However, as it is trivial to validate if a transmitted bid has been sent within the correct time interval, the auctioneer can detect such cases on the server-side and, e.g., disqualify the user from further participation in the auction.

3. DECOUPLING STRATEGIES

This section provides our first contribution: decoupling strategies and simulations of their expected influence on the clients' transmission rate. A decoupling strategy specifies when and how long a placed bid is held back on the local client. In this section we consider *open-loop control* [10], where fixed input parameters are used for the decoupling strategies. In later sections we extend this open-loop approach by providing *closed-loop* control, which extends open-loop control with (i) a distributed feedback channel (control channel) and (ii) a PID controller to dynamically adapt the input parameters to attain an optimal server utilization.

Each of the examined rate control strategies can be used in two different variants: *Data-overwrite* and *data-queue*. In the *data-queue* variant, all submitted data are queued and eventually transmitted to the server, while in the *data-overwrite* variant, items not yet sent are overwritten by subsequent items—only one item per-auction is queued at a time. The *data-overwrite* strategy is applicable to all temporally decoupled systems where later information obsoletes former information, such as bids in a first-price sealed-bid auction system.

3.1 Group-based control

With *group-based* rate control we partition the set of clients into disjoint subsets, where at a given point in time only the clients within a particular subset are allowed to transmit data to the server. There are two distinct steps: (i) Partitioning the original set into a set of disjoint groups and (ii) the selection of an active group. In our approach these two concepts are intertwined: Each client has a unique ID. When the server wants to select a particular group it broadcasts two values: A *divisor* and a *modulo* value. The divisor specifies how many groups exist, while the modulo value specifies which of these groups is currently active. Each client divides its ID by the divisor and verifies if the congruence class modulo the divisor matches the *modulo* value. If this is the case, the client is within the *active* group. Otherwise, the client needs to wait until its group gets activated.

The modulo value is incremented in predefined time intervals, enabling iteration over existing groups. This interval between groups in combination with the amount of groups determines the maximum possible delay of client requests. In the worst case a client wants to issue a request right at the moment where it became *inactive*. Thus, the client has to wait $interval\_between\_groups \cdot (amount\_of\_groups - 1)$ time units until it is able to transmit the request.

3.2 Interval-based rate control

Interval-based rate control exploits the fact that during a peak not only the overall system load increases, but also the transmission rate of individual clients. The idea is to partition time into disjoint intervals with length i and to allow each client to send at most one data item during such an interval. If a client has not transmitted any data to the server in a particular interval, it can transmit the next data item directly to the server without any delay. Otherwise, the client needs to wait for the next interval to be able to send further data. It is crucial that the start points of the intervals differ between clients. Otherwise, all clients with queued packets that wait for the next interval would start submitting their queued packets at exactly the same time.

An advantage of the interval-based approach is that it does not delay data transmission under low load, i.e, when new data items are created at a client in time intervals larger than the submission interval. Only in cases where more than one data item per interval is created, transmission of data will be delayed. This approach can be seen as a special case of token bucket based rate control with a bucket size of one and a new token added each i seconds. To control the request rate of clients, the size of the interval i can be adapted.

3.3 Simulation

In this section we use discrete event simulation [8] to validate the performance benefit of our temporal decoupling approaches. Compared to traditional simulation approaches, discrete event simulation allows for efficient simulation of long time spans within short amounts of time. The simulation uses a priority queue of events sorted after the events' time from which events are iteratively dequeued and processed. When an event is executed, the simulated clock is updated to the value of the event's time. The execution of an event can lead to additional events that are pushed on the priority queue.

First, we generate a load curve that represents when bids are submitted by users during the peak load. Afterwards, we examine the effectiveness of our two temporal decoupling strategies. For each of the strategies we provide performance evaluations for different parameters and both variants (*data-overwriting* and *data-queuing*). Neither the shape of the curves nor the relative load at a particular point in time depend on the total number of clients. Therefore, we give the load relative to the peak load that occurs when no decoupling strategy is used.

3.3.1 Bid behavior

For the simulation of the bidder's behavior we focus on the area around the peak load. Using statistical data obtained from real world auctions of our industrial project partner we observe that $\frac{2}{3}$ of the bids are placed during the peak load in the final five minutes of an auction. We exploit the fact that in a Gaussian distribution 62.27% of all measurements are located within a distance of σ from μ. As these 62.27% roughly correspond to the $\frac{2}{3}$ of the bids, we can assume a value of 150 seconds for σ, to let the Gaussian distribution approximate a peak load over a duration of five minutes. For μ we use a value of zero—placing the peak of the load at the null point of the diagram.

The resulting load curve representing the bidding behavior is shown as grey line in the diagrams in Figure 2. The x-axis represents the time in seconds and the y-axis represents the amount of placed bids per second given in percentage of the peak load of the original load curve. This load curve is used as input for our two different decoupling strategies.

3.3.2 Group-based rate control

For the simulation of the *group-based* strategies we simulated the load with two different sets of input parameters: In the first case we used 20 groups—shown in Figure 2(a)—while in the second case we used 75 groups—shown in Figure 2(b). In both cases we switch groups in 4 second intervals. In addition, we simulated the *data-overwriting* behavior (in red with dashed linestyle) and the *data-queuing* behavior (in blue with solid linestyle). The parameters used in the simulation have been chosen according to the constraints—such as the maximum time span between the deadline for bids and the deadline for messages—in real-world first-price sealed-bid auctions.

First, we observe that there are no considerable differences between the original load curve and the decoupled load curves when *data-queuing* is used. The only difference is that in the case of *data-queuing* the load curve is slightly shifted to the right, which is the expected behavior, as the total amount of transmitted bids does not differ and as bid submission is never delayed for more then one full iteration of all groups. However, when *data-overwriting* is used, significant load reductions can be observed. Again, this is the expected behavior, as at most one bid is queued while the node itself is not active, even if multiple bids are placed. Rate reduction with *data-overwriting* is only effective, if—on average—more than one bid is placed on a client while it is inactive, as otherwise the resulting rate would be the same as in the data-queuing scenario.

3.3.3 Interval-based rate control

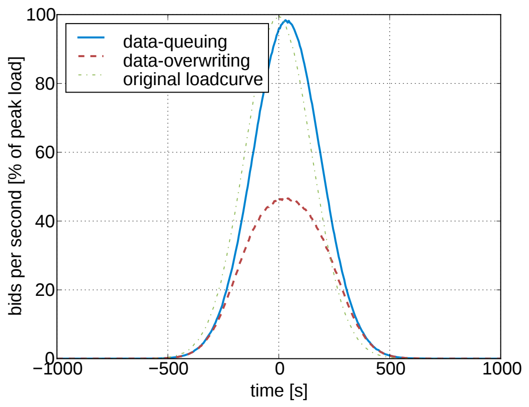
Similarly to the previous strategy, we simulated the results for two different parameters. In the first case we used an interval size of 80 seconds (shown in Figure 2(c)), allowing only one bid to be transmitted within an interval of 80 seconds. Bids that cannot be placed immediately are delayed and placed after the interval's end. In the second case we used an interval size of 300 seconds (shown in Figure 2(d)). In both cases we simulated the *data-overwriting* and the *data-queuing* variants. As in the case of group-based rate control the parameters have been chosen according to the constraints of real-world first-price sealed-bid auctions. Therefore, the maximum possible delay in the two interval-based data-overwriting cases corresponds to the maximum possible delay in the two group-based data-overwriting cases.

In the results we can observe a brief overshoot and then a relatively stable amount of load. This is the expected behavior: The overshoot occurs as during the increasing overall bid rates some clients have not yet transmitted a bid in the current interval. Thus, they can immediately transmit a newly placed bid without any rate restrictions. Once the interval-based rate control sets in, this is not possible as clients typically transmit bids in each individual interval.

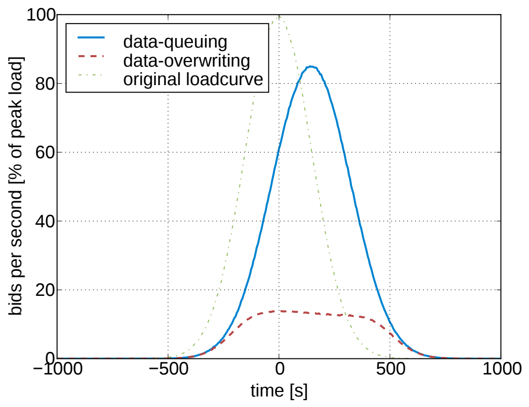
In the *data-overwriting* variant, the height and the width of the load curve are comparable to the *group-based strategy* examined in the previous section. However, in the *data-queuing* variant, there is a fundamental difference between the *group-based strategy* and the *interval limitation* strategy: While in the group-based strategy *data-queuing* leads to a higher load when compared to the *data-overwriting* case, in the *interval limitation* strategy, *data-queuing* does not show a higher peak load, but an increased duration. Again, this is the expected behavior, as interval-based rate control restricts the possible transmission rate. Thus, in *data-queuing* the larger amount of bids leads to additional transmission delays. With an interval size of 80 seconds, we can observe that the original curve would have a peak at around 28 000 bids per second, while the decoupled curve shows a peak at around 14 000 bids per second—which essentially doubles the amount of users a system can handle. With larger delays or intervals the peak load can be even further decreased. However, larger interval sizes also imply larger time spans between the deadline for bids and the earliest possible deadline for messages.

3.3.4 Transmission failures

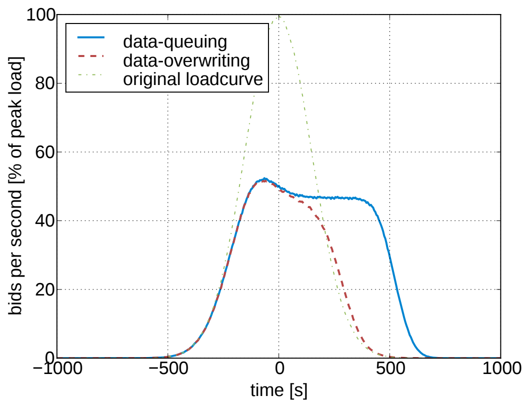
Figure 3 shows the performance of interval-based rate control in case of transmission failures. In the depicted scenario, bids cannot be transmitted to the server between the time values 0 and 180 on the x-axis. For the simulation an interval size of 80 is used. The *re-transmission* strategy of a client is to try again when the next local interval starts. In the *data-overwriting* variant the only difference to Figure 2(c) is that the transmission rate drops to zero during the outage. As each client can have at most one outstanding bid, the point until all bids have been received at the server is not substantially deferred. In the *data-queuing* variant each single bid needs to be transferred to the server. Therefore, the duration until all bids have been collected at the server is prolonged by approximately the duration of the outage. Due to the re-transmission strategy, the transmission rates



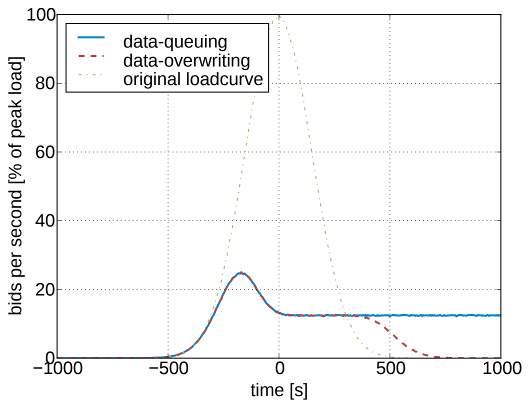
(a) Group strategy: 20 groups, iteration each 4 seconds



(b) Group strategy: 75 groups, iteration each 4 seconds



(c) Interval limit strategy: 80 seconds



(d) Interval limit strategy: 300 seconds

Figure 2: Group-based and interval-based decoupling strategies

of clients do not change when the server is not available. In the figure this can be observed by the server's load instantly reaching its *normal* value as soon as transmission is possible again.

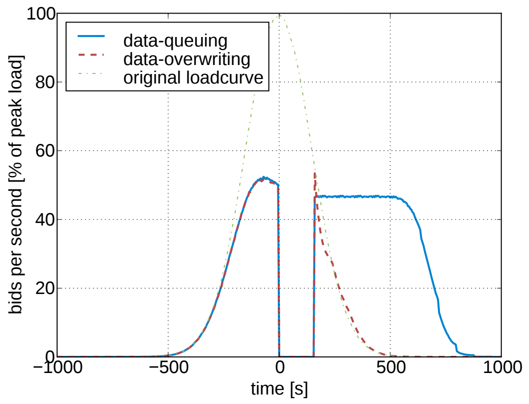


Figure 3: Transmission failure mitigation

3.3.5 Results

Our results show that the group-based and interval-based decoupling strategies are suitable for temporal decoupling to mitigate temporary peak loads. The *group-based* strategy allows to mitigate peak loads by assigning nodes to a set of groups and allowing nodes to transmit data only when their particular group is active. We have evaluated this strategy with two different variants: *Data-queuing* and *data-overwriting*. As bids that are placed while the node's respective group is inactive are delayed, both variants also slightly shift the load curve to the right. However, for a given delay parameter the data-overwriting variant is able to reach a considerably lower peak load then the data-queuing variant.

The idea behind the *interval limit* strategy is to limit the maximum transmission rate of each single client. Therefore, from a bidder's point of view bids are not delayed until the local transmission rate exceeds the allowed transmission rate. For both variants—*data-queuing* and *data-overwriting*—the resulting peak load is the same for a given parameter. However, the time until all bids have been collected at the server is longer when *data-queuing* is used, due to the larger amount of total bids.

The simulations in this section used predefined input parameters, such as the amount of groups or the interval size. While those input parameters allow to reduce request rates, they are open-loop control approaches that do not take actual load at the server into account. Thus, these approaches are not able to prevent server overload if input parameters are suboptimally chosen, e.g., if the amount of active clients changes over time. Moreover, open-loop control only achieves a sub-optimal server utilization. In the next section we upgrade this open-loop approach to a closed-loop approach, allowing for an optimal server utilization.

4. CLOSED-LOOP CONTROL

In this section we contribute with our closed-loop control approach that integrates our decoupling strategies with (i) a distributed feedback channel and (ii) a PID controller to induce an optimal utilization at the server. The PID controller dynamically adapts the input parameters of our decoupling strategies—in the context of the controller labeled as *manipulated variable* (MV)—while the distributed feedback channel can relay control information to the clients with only minimal overhead at the server.

The advantage of closed-loop control in comparison to the open-loop strategies presented in Section 3 is that the PID controller allows the system to adapt to changing transmission rates of clients, as well as a changing number of overall clients. While open-loop decoupling strategies can adapt to the worst-case considering the highest potential overall server load, this unnecessarily delays bid transmission at the clients. Consequently, malicious bidders would have more time to tamper with the stored bids, thereby increasing the system's security threats.

System and software architectures are depicted and described in Figure 4. First, we discuss the setup at the client-side. Then we proceed with a discussion of the controller. Finally, we conclude the section with our distributed feedback channel.

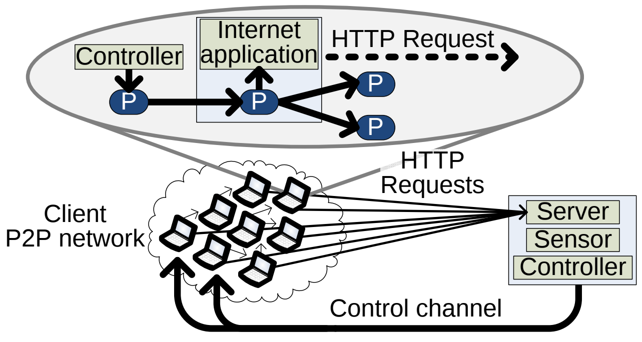


Figure 4: Control loop architecture overview. The individual clients send requests to the server. At the server the sensor measures the server-side queuing and processing delay and the controller uses the input provided by the sensor to adjust the output, e.g., the interval length or group size of our decoupling strategies. A distributed feedback channel is used to scalably propagate the control output back to the clients. On each client the Internet application originates the application-level requests, while the client application (labeled *P*) receives input from the controller over the distributed feedback channel and provides input to other peers and the Internet application.

4.1 Clients

On the client-side we use a decoupling client to facilitate request rate control. This decoupling client performs two tasks: (i) It acts as actuator by receiving control information over the distributed feedback channel and by controlling request rates of the Internet application accordingly, and (ii) it is itself part of the distributed feedback channel as it forwards control information to other clients. Due to our closed user group, installation of custom client-side software is a feasible approach. As a consequence, we do not target generic Internet applications, but specific types of applications such as first-price sealed-bid auctions that benefit from temporal decoupling as outlined in Section 2.2.

4.2 Controller

The task of the controller discussed in this section is to adaptively control the parameters of the rate control techniques introduced in Section 3. We use a *closed-loop* control approach that allows the controller to provide feedback to individual clients and thereby provides a better performance in case of unpredictable loads and unanticipated load changes.

The setup of our controller is depicted in Figure 5. The controller is implemented as PID controller, as measurements showed that it provides a lower standard deviation of the *process variable* (PV) than a P (Proportional) or PI (Proportional-Integral) controller in our application scenario. Clients regularly send requests to the server. The input to the controller is provided by a sensor at the server that measures the *process variable*. The controller then compares the *error* (or deviation) e between the given *setpoint* (SP) and the measured process variable. The goal of the controller is to minimize e by adapting the input parameters of the system accordingly. In our system the process variable is—depending on the configuration—either the queue length of requests waiting to be processed, or—alternatively—the request processing delay at the server. Consequently, the setpoint is the target value of the system and given in the same unit as the process variable. The *manipulated variable* (MV) is the input to the process, in our system this corresponds to the input parameters of the decoupling mechanisms introduced in Section 3. In case of group-based decoupling this parameter reflects the percentage of active clients, while in the case of interval-based decoupling the parameter reflects the interval size.

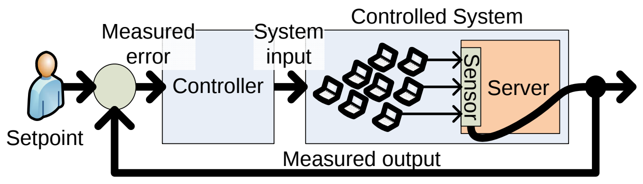


Figure 5: Control loop. Our controlled system consists of the individual clients and the server. A sensor at the server measures the processing delay. The measured error between an externally provided setpoint and the measured output is fed into the controller, which adjusts the system input accordingly. The system input is provided via a distributed feedback channel back to the clients.

Due to the fact that the system behavior differs between application scenarios and as we anticipate that in real-world deployments at best a heuristic tuning or auto-tuning method will be used, we use the Ziegler-Nichols heuristic method [6, 28] to obtain P, I, and D gains with a quality that can be compared to such real-world deployments. First, the P gain is increased until the system starts to oscillate. Then, the Ziegler-Nichols charts are used to calculate the respective gains for a PID controller.

4.3 Distributed feedback channel

The task of the distributed feedback channel is to transmit control information from the controller to the clients. The typical implementation approach is to let the server communicate with each individual client. However, previous work has shown that for large amounts of clients such a system cannot be easily implemented without a specialized broadcasting infrastructure [7].

Consequently, we contribute with the integration of a broadcasting infrastructure based on an overlay network that can be used to *flood* information to the set of clients. We use application-level broadcast [27] based on a tree-based network structure as depicted in Figure 6. Therefore the height of the tree—and thereby the propagation delay—grows only logarithmically with the amount of clients. To prevent failures caused by single clients, we use multiple independent trees, so that each client receives broadcasts from multiple sources. To detect partitions we use a keep-alive mechanism. The server regularly broadcasts cryptographically secured sequence numbers to the tree. A client can detect problems, if it has not received sequence numbers for some time or if individual sequence numbers are missing. However, due to the closed user group in our application scenario we expect a low node churn.

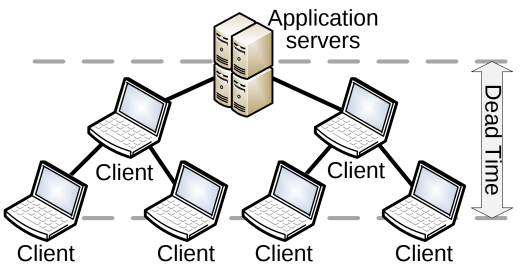


Figure 6: Distributed feedback channel. The servers send information to only a few clients with a tree-based network structure used to propagate data between clients. Parameters of the tree are the arity and the height.

For our distributed broadcast channel there is a trade-off between the total number of clients and the stability of our system:

- With a growing number of clients the height and/or arity of the tree have to increase to accommodate more clients. However, with a growing height or arity the dead time of our system increases as well, which in turn leads to instabilities such as oscillation.

- With increasing height of the tree, the clients' average distance to the server increases. Thus, it takes longer until a server broadcast is received by a majority of the clients. Similarly, with increasing arity each client has to transmit information to a larger amount of children, lowering the bandwidth available for each individual transmission.

- When the dead time of the system increases, this decreases the stability, making the system susceptible to overshoots or oscillation. It is possible to mitigate this issue by decreasing the gains of the controller. However, this slows down the system, potentially preventing it from responding to load changes in time.

By adapting the arity and the height of the N-ary tree, we can control the trade-off between the maximum amount of clients and the dead time. For example, consider an average propagation delay between two clients of 100 ms and a maximum tolerable dead time of 5 seconds. This would allow a total tree height of 50. When using a binary tree, this equals a theoretic total of $22.5 \cdot 10^{14}$ clients. In practice, the number of clients is therefore not limited by our broadcast channel, but by the lower bound of the request rate acceptable for each individual client. Under most configurations, the feedback channel itself is capable of supporting an amount of clients several dimensions higher than the maximum amount reasonably supported by the servers of the system. As a consequence, we can use the additional capacity to increase the redundancy of our tree, allowing for correct propagation of broadcasts in cases where individual clients fail.

As a majority of bidders generally places bids shortly before the auction deadline and as the set of bidders is predefined for each auction, node churn during this critical period is typically smaller than in comparable application scenarios—if not non-existent at all. Therefore, our distributed feedback channel is not specifically adapted to high node-churn scenarios. However—depending on the concrete scenario—alternative application-level broadcast protocols can be used for such cases.

5. PERFORMANCE MEASUREMENTS

In this section we evaluate the effectiveness of our rate control approach. The goal of the measurements is to evaluate different system configurations in regard to stability and performance. While there are several common benchmarks for evaluating Web applications, they are not applicable to our temporally decoupled system. RUBiS (Rice University Bidding System) [3] models an auction site similar to eBay. While in RUBiS the typical user interactions such as browsing of auctions and consulting of bid histories are modeled, in our system the focus lies on the request rate control of temporally decoupled bid submissions. Similarly to RUBiS, TPC-W [24] models a Web shop to test the performance of Web server and database systems. As a consequence, neither RUBiS nor TPC-W can be used in their current specification to evaluate our rate control system, as they are not applicable to temporally decoupled systems. Instead, we evaluate our system in the auction scenario specified in Section 5.4.

In the following evaluations we first examine the open-loop behavior of our system and the system's response to changes in the input. Then, we examine the closed-loop behavior of our controller in a simulated auction scenario using group-based and interval-based control. In addition, we also compare our approach to alternative solution approaches, such as an approach piggybacking control information in HTTP requests and another approach rejecting requests on the server-side [1].

5.1 Configuration

In the evaluation we focus on single-threaded clients using *data-queue* for transmission and the processing and queuing delay as metric. In addition, we also conducted measurements where we limited the number of concurrently processed requests at the server to 1, and subsequently used the size of an unbounded queue of the waiting requests as a metric for the controller. Such a setup better reflects application servers with a fixed number of worker threads. However, as we did not observe significant differences in comparison to the processing and queuing delay metric, we only show the results for the processing and queuing delay metric in this section, which also reflect our results for the queuing size metric.

5.2 Test setup

For the evaluation, we measure how our controller works for a CPU bound service, as this has been indicated by our industrial partner to be the limiting factor. Our service performs a simple calculation—prime factorization—that requires about 100 ms of processing time. An increasing concurrency rate also increases the processing delay for each individual request. The hardware setup consists of six standard PCs (Personal Computers) running on Ubuntu 8.04 server edition. The first PC hosts the server, while the remaining PCs act as load generators. The server is implemented as Java application using Jetty to provide HTTP access. The clients are also implemented in Java with one thread per simulated client. Each load generator is responsible for the simulation of multiple clients. Due to the fact that we test a CPU bound service, local network performance is not an issue.

As all clients run on the same local network and as multiple clients run on the same host, the propagation delay between individual clients is low. To account for the fact that dead time has a considerable influence on the performance of a controlled system, we implemented *dead time simulation* into the distributed broadcast channel used to propagate control information to clients. This simulation works by enabling a client to hold feedback information for a particular amount of time before forwarding this information to the next client. This allows to simulate different network conditions, such as latency on wide area networks (WANs).

5.3 Response to request change

First, we evaluate how fast our system reacts to sudden changes of the request rate. The evaluation was done in two distinct steps depicted in Figure 7.

In the "No Control" curve the behavior of the uncontrolled system is shown and in the "PID Controller" curve our system is extended with a PID controller that provides closed-loop control. In both cases we double the amount of active clients and verify how the system reacts to this input change. In the "No Control" case, doubling the amount of clients also doubles the processing time at the server. In the "PID Controller" case, the controller tries to keep the response time stable at 1 000 ms. We can see a peak at the beginning, when the controller needs to find an optimal range at the start of the simulation, and during the change, when the controller needs to adapt group sizes to the new client sizes accordingly (see circles).

Stability of the controller is a considerable factor, as it

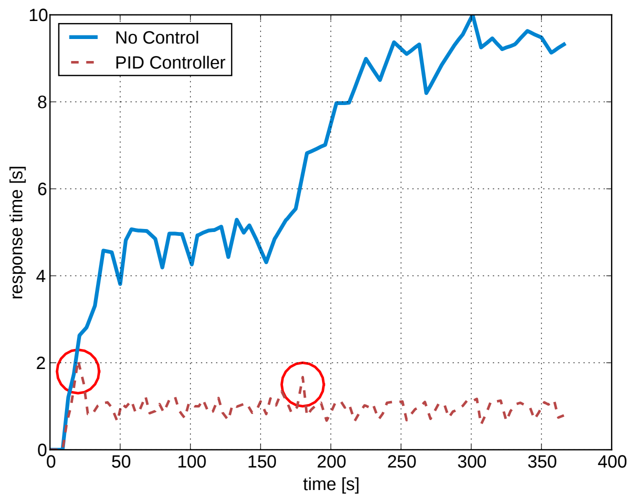


Figure 7: Response to system change: Without controller and with PID controller in place

does not only affect the processing time at the server, but also the total throughput of the system. With a low stability caused by an unstable PID controller, there are time periods where no clients send requests, although the server is not fully loaded. In addition, also the case where too many requests are sent by clients can cause decreased throughput due to overload at the server. Generally, the goal of a controller is to reach the best performance possible with a given limit for the stability.

5.4 Auction scenario

For the second test we examine the behavior of our controller when used for the characteristic load of our auction scenario. In particular, we examine the performance during the peak load that resembles the bids of a first-price sealed-bid auction [13].

To model the peak load we distribute each client's request rate according to a Gaussian distribution with $\sigma = 50$ seconds and $\mu = 130$ seconds. There is a total of 1 000 clients and each client transmits a total of 30 requests. For each request at the server, the server executes a calculation that takes 100 ms. The set-point of the controller that reflects the acceptable processing delay is set to 1 000 ms.

The results of this evaluation for group-based rate control are presented in Figure 8. In the first measurement we examine the response time when no rate control is used, i.e., not even open-loop. The maximum response time peaks at around 10 000 ms. The reason for the plateau at 10 000 ms is given by the fact that the concurrency of each client is limited to 1 and that thus for 100 concurrent requests for an operation that takes 100 ms the total average will not exceed 10 000 ms. In the next three measurements we examine the behavior of our PID controller for different amounts of dead time.

In the first case, the simulated dead time is near zero. The actual dead time is non-zero due to inevitable delays in our broadcast channel. In the second case we simulate a dead time of 500 ms per level of the broadcast tree and in the third case a dead time of 2 000 ms. In each of the cases there is a total of 3 levels in a quinary (5-ary) tree. While the con-

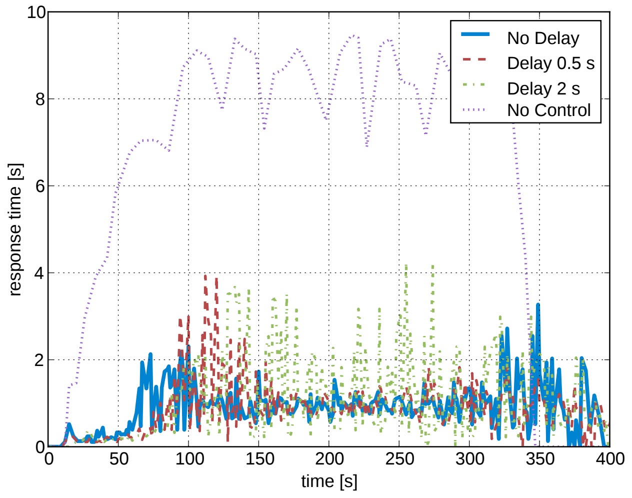


Figure 8: Auction scenario with: PID controller, group-based control, average process delay metric

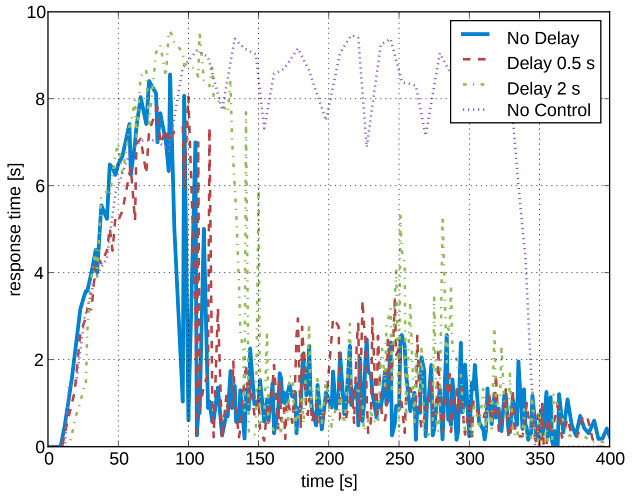


Figure 9: Auction scenario with: PID controller, interval-based control, average process delay metric

troller is able to constrain the processing delay in the first case, in the second case it takes some time until the system is stable. In the third case we see a relatively large oscillation during the whole test, suggesting that the dead time is too high for our controller. In a real-world application scenario such propagation delays between nodes are considerably lower. If we assume an propagation delay of 50 ms, a total of 30 levels would yield a maximum propagation delay of 1500 ms, comparable to the maximum propagation delay of the simulation with a dead time of 500 ms. A quinary tree with 30 levels would support a theoretical maximum of $1.16 \cdot 10^{21}$ nodes, which is more than anyone would need in practice.

During the evaluation of group-based rate control we observed that the limited granularity of possible group sizes can have negative impact on the performance of the system. Especially when large groups are used, the possible group sizes that can be declared using a single divisor and a single modulo value often considerably differ from the manipulated variable calculated by the controller. As mitigation strategy, we enhanced group-based control to use lists of divisors and modulo values, instead of two single values. Clients matching a divisor/modulo combination in the list are allowed to transmit information. This allows for a more fine-grained specification of group-sizes, and thereby for a better stability of the controller.

In Figure 9 we show the results for interval based rate control. While the output of interval-based control is stable after the initial peak, variations are slightly larger than with group-based rate control. The stable behavior in Figure 9 is achieved by limiting the maximum difference between two consecutive rate control parameters calculated and broadcast by the controller. The respective limit is automatically derived based on the estimated amount of clients and the measured average time per operation. Due to this limitation the controller requires some time until it can reduce the load during steeply increasing client request rates.

5.5 Comparison

In this section we compare the effectiveness of our approach with two alternative approaches found in related work (Section 7) that work without distributed feedback channels.

In the first approach in Figure 10 we use a *piggyback* strategy where we embed rate control feedback in HTTP responses. When an HTTP client sends an HTTP request to a server, the server includes rate control information in the HTTP response. The results show that the piggyback approach is unstable and leads to oscillation. A major cause for this problem are variable dead times and the fact that control information at clients cannot be updated between individual requests of a client. Thus, the system is only able to provide request rate control, but not admission control as our distributed feedback channel.

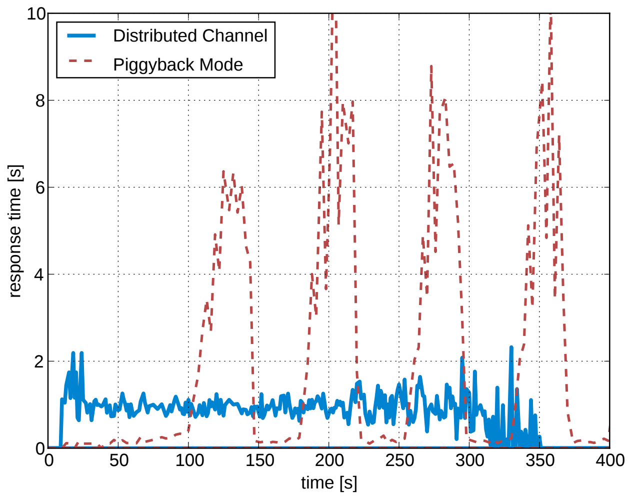


Figure 10: Comparison between distributed feedback channel and piggyback approach

In the second approach in Figure 11 we use a *reject* strategy similar to the control-theoretic approach by Abdelzaher et al. [1] where we reject requests at the server instead of the client. When a client is not able to establish a connection to the server, we use an exponential backoff strategy similar to TCP's retransmission timer [17] without the adjustments using the Smoothed Round Trip Time (SRTT) [19]. The original approach is not directly applicable to our application scenario, as implementation of a *reject/request-ratio* would not allow us to reject requests on a per-client basis. Instead, we reject requests at HTTP level instead of TCP level, as our existing decoupling mechanism depends on the client IDs. As a consequence, the performance of the *reject* approach in our evaluation is restricted by our required adaptations.

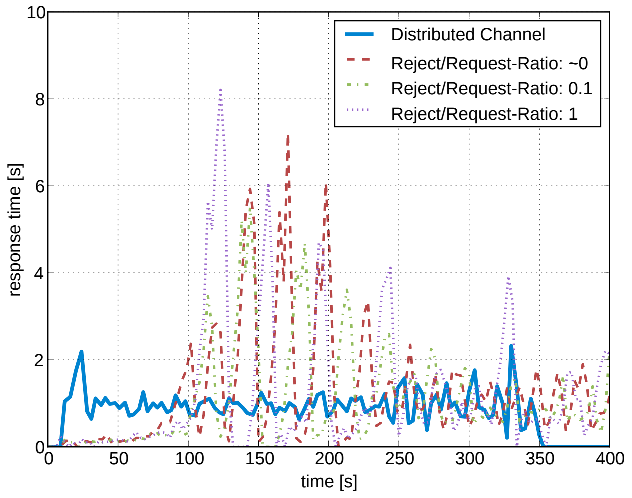


Figure 11: Evaluation of different reject/request-ratios

With the *reject* strategy the results depend on the ratio between the cost of accepting a request and the cost of rejecting a request. Figure 11 depicts the results for different ratios between accepting requests and rejecting requests. In the first case processing a request takes 100 ms, while we do not artificially delay the time for rejecting a request. In the additional two cases we increase the time for rejecting a request to 10 ms and 100 ms to simulate a reject/request-ratio of 0.1 and 1 respectively. While the results show that the stability does not significantly differ between different reject/request-ratios, there is a considerable difference in the throughput. While our distributed feedback channel is able to achieve a throughput of 8.6 requests per second, the throughput in the best *reject* approach is only 7.0 requests per second. When increasing the reject/request-ratio the throughput declines further to 5.0 and 2.9 requests per second accordingly. Therefore, the benefits of our distributed feedback channel are primarily relevant in scenarios with a high reject/request-ratio, where the cost of rejecting a request is not negligible. Abdelzaher et al. have shown that this is the case in Web-server end systems, where they measured a reject overhead of 1.1 ms per request, while processing a request for a zero-sized URL took 1.604 ms [1]. A more detailed examination of the reject/request-ratio's influence is given in the next section.

6. LIMITATIONS OF OUR APPROACH

The applicability of our approach for certain application scenarios can be determined by comparing the effort required at the server for processing requests with the effort required for rejecting requests and with the typical ratio between average load and peak load. For comparison, we assume a simple system that does not use distributed load control: If a request cannot be processed, it is rejected by the system and the client does not try to retransmit the request.

First, we examine the theoretical limitations due to the cost difference between accepting and rejecting requests. We then proceed by discussing the implications of these trade-offs on real-world application scenarios.

6.1 Theoretical limitations

We denote the maximum request rate that the system can sustain without rejecting requests as $r_{process}$, the factor between the system load under normal operation and the system load during a peak as $factor_{peak}$, and the factor between the cost of accepting a request and rejecting a request as $factor_{request}$ $(= \frac{cost\ for\ request}{cost\ for\ reject})$. We then calculate an upper bound for the server's effort of rejecting requests when not all requests can be processed by calculating $r_{process} \cdot \frac{factor_{peak}}{factor_{request}}$: We have $factor_{peak}$ as many requests, but rejecting each request only takes $\frac{1}{factor_{request}}$ of the time it would take to process these requests.

If $factor_{peak}$ is equal to $factor_{request}$ the system is not able to handle any business requests during peak load, as rejecting requests takes up all available resources. Thus, in this case our distributed broadcast channel would be able to double the effective capacity of the system, as the resources previously used for load control can now be used for request processing. With a higher $factor_{request}$ the advantage of our distributed broadcast channel increases, while with a lower $factor_{request}$ the advantage decreases. For example, if $factor_{request}$ is 100, but $factor_{peak}$ is only 10, the upper bound of rejection costs during the peak load is about $\frac{1}{10}$ of the system's overall performance. In this case, the advantage of *outsourcing* the broadcast channel is rather limited.

6.2 Real-world applications

In a real world system the examination of our system's advantages are more complex than the calculations for the simplified model given in Section 6.1: (i) Our model does not deal with retransmission of requests. If a request is rejected, the client would need to try a retransmission at some point. This increases the advantage of our approach. (ii) We assume that each request is transmitted independent of other requests. In reality, a client can use information from earlier requests to optimize the rate for later requests. This in turn decreases the advantage of our approach, but may lead to unstable behavior as shown in Figure 10.

Furthermore, while our approach cannot prevent clients from *trying* to cheat by ignoring control information, the server can easily verify if individual clients act according to the broadcast control information. If clients violate those rules, the server can ban those clients from further participation in the auction by rejecting requests without processing them.

7. RELATED WORK

Our system is not the first effort to dynamically control request rates of clients on higher level application layers. For example, several papers [1,11,18] describe approaches that use controllers to react to system load and thereby manage to improve the system's overall performance. The main difference to our work is that we do not take actions on the server side (such as rejecting requests or content adaption), but rather try to solve the problem directly at the source—at the client side. This section gives a brief overview of relevant and related work. First, we discuss load control mechanisms in standard Internet protocols. Afterwards, we proceed to specialized solutions for particular types of Web applications.

Transmission Control Protocol.

TCP based servers [19] are able to implicitly control the transmission rate of clients by only allowing a particular amount of parallel connections and by using a *window* to restrict the amount of data waiting to be processed. If SYN packets of clients are dropped, clients use an exponential backoff approach [16,17] to time retransmissions, which can further be improved with a PI controller [12]. In our application scenario, TCP based rate control is not fully applicable as our goal is to prevent overload by filtering requests directly at the clients.

Control-theoretic approaches.

Abdelzaher et al. [1] describe performance control of a Web server using classical feedback control theory. Another control-theoretic approach guaranteeing bounded and predictive response times by Web servers been based published by Lim et al. [11]. Chan et al. [4] propose a fuzzy PI controller to guarantee proportional delay differentiation on Web servers by providing a better service to a premium class of users. The main difference to the first two works is that we directly control the request rates at clients. The work of Chan et al. could complement our solution.

Self-adapting service level.

Philippe et al. describe an approach for an self-adapting service level [18] that allows some components in an application server to dynamically degrade or upgrade their level of service, thereby trading a lower service level for a better overall performance of the server. This could complement our solution in general, but is not applicable for our specific application scenario.

In addition to the discussed publications there is a number of publications [2,5,9,14,22,25,26] that deal with admission control for Web server systems. Most existing systems adapt parameters at the server to influence the load produced by clients, e.g., by deciding which requests should be rejected. In contrast, our technique frees the server from rejecting individual requests, as we directly filter requests at the client.

8. CONCLUSION

This paper presents a new approach for adaptive load control and performance for first-price sealed-bid auctions. Our main contribution is the integration of (i) a distributed feedback channel to transmit control information from the server to the clients with (ii) decoupling strategies that allow to constrain client requests directly at the client side and (iii) a PID controller that adaptively controls the input parameters of those decoupling strategies to facilitate an optimal server utilization.

In comparison to established techniques, our system allows to control the quality of higher layer protocols without relying only on the load control mechanisms provided by lower layer protocols. Unlike related work, the servers in our system do not need to communicate with each single client individually. In particular, we can control request rates of clients even before a connection to a server is established, leading to a significant reduction in the required server-side resources. Our system does not only prevent clients from overwhelming the capacity of the servers, but allows to reduce the capacity required at the server-side infrastructure for applications that show temporary peaks in transmission rate or that can queue and send client requests in a single batch request, whereby data obsoleted by newer information can already be filtered at the client side.

Concluding, our approach provides viable means to manage high loads in first-price sealed-bid auctions without requiring large clusters able to cope with brief peak loads. Controlling transmission rates at clients decreases the number of required servers and makes more efficient use of available resources. A potential drawback is the necessity of a distributed feedback channel, which, however, could be approximated through similar infrastructures. As future work we identify the use of a Smith predictor [21,29] to compensate for dead time due to delays and thereby stabilize the controller as well as an evaluation of fuzzy controllers [4] to verify if they are able to yield acceptable results in our system.

9. REFERENCES

[1] T. F. Abdelzaher, K. G. Shin, and N. T. Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. *IEEE Trans. Parallel Distrib. Syst.*, 13(1):80–96, 2002.

[2] M. Andersson, J. Cao, M. Kihl, and C. Nyberg. Admission control with service level agreements for a web server. In M. H. Hamza, editor, *EuroIMSA*, pages 275–280. IASTED/ACTA Press, 2005.

[3] E. Cecchet, J. Marguerite, and W. Zwaenepoel. Performance and scalability of ejb applications. In *OOPSLA*, pages 246–261, 2002.

[4] K. H. Chan and X. Chu. Design of a fuzzy PI controller to guarantee proportional delay differentiation on web servers. In M.-Y. Kao and X.-Y. Li, editors, *AAIM*, volume 4508 of *Lecture Notes in Computer Science*, pages 389–398. Springer, 2007.

[5] D. Dyachuk and R. Deters. Transparent admission control and scheduling of e-commerce web services. In J. Filipe and J. A. M. Cordeiro, editors, *WEBIST (Selected Papers)*, volume 8 of *Lecture Notes in Business Information Processing*, pages 124–136. Springer, 2007.

[6] T. Hägglund and K. J. Åström. Revisiting the Ziegler-Nichols tuning rules for PI control. *Asian Journal of Control*, 4(4):364–380, Dec. 2002.

[7] M. Hauswirth and M. Jazayeri. A component and communication model for push systems. In O. Nierstrasz and M. Lemoine, editors, *ESEC / SIGSOFT FSE*, volume 1687 of *Lecture Notes in Computer Science*, pages 20–38. Springer, 1999.

[8] J. O. Henriksen, R. M. O'Keefe, C. D. Pegden, R. G.

Sargent, and B. W. Unger. Implementations of time (panel). In D. W. Jones, editor, *WSC '86: Proceedings of the 18th conference on Winter simulation*, pages 409–416, New York, NY, USA, 1986. ACM.

[9] M. Kihl, A. Robertsson, A. Andersson, and B. Wittenmark. Control-theoretic analysis of admission control mechanisms for web server systems. *World Wide Web*, 11(1):93–116, 2008.

[10] M. M. Kokar, K. Baclawski, and Y. A. Eracar. Control theory-based foundations of self-controlling software. *IEEE Intelligent Systems*, 14(3):37–45, 1999.

[11] S. S. Lim, C. Lee, C. W. Ahn, C. G. Lee, and K. H. Park. An adaptive admission control mechanism for a cluster-based web server system. In *IPDPS*. IEEE Computer Society, 2002.

[12] C.-H. Lung and O. W. W. Yang. Evaluation of an adaptive PI rate controller for congestion control in wireless ad-hoc/sensor networks. In *CSE (2)*, pages 597–602. IEEE Computer Society, 2009.

[13] R. P. McAfee and J. McMillan. Auctions and bidding. *Journal of Economic Literature*, 25(2):699–738, June 1987.

[14] P. J. Meulenhoff, D. R. Ostendorf, M. Zivkovic, H. B. Meeuwissen, and B. M. M. Gijsen. Intelligent overload control for composite web services. In L. Baresi, C.-H. Chi, and J. Suzuki, editors, *ICSOC/ServiceWave*, volume 5900 of *Lecture Notes in Computer Science*, pages 34–49, 2009.

[15] S. Mogaki, M. Kamada, T. Yonekura, S. Okamoto, Y. Ohtaki, and M. B. I. Reaz. Time-stamp service makes real-time gaming cheat-free. In G. J. Armitage, editor, *NETGAMES*, pages 135–138. ACM, 2007.

[16] A. Mondal and A. Kuzmanovic. Removing exponential backoff from TCP. *Computer Communication Review*, 38(5):17–28, 2008.

[17] V. Paxson and M. Allman. Computing TCP's Retransmission Timer. RFC 2988 (Proposed Standard), Nov. 2000.

[18] J. Philippe, N. D. Palma, F. Boyer, and O. Gruber. Self-adapting service level in java enterprise edition. In *Middleware 2009*, volume 5896 of *Lecture Notes in Computer Science*, pages 143–162. Springer Berlin / Heidelberg, 2009.

[19] J. Postel. Transmission Control Protocol. RFC 793 (Standard), Sept. 1981. Updated by RFCs 1122, 3168.

[20] Selfcite, 2008. Fast abstract—Self-reference removed due to double-blind review.

[21] P. Sourdille and A. O'Dwyer. A new modified smith predictor design. In *ISICT*, volume 49 of *ACM International Conference Proceeding Series*, pages 385–390. Trinity College Dublin, 2003.

[22] P. B. Srinivas, S. Ramanathan, and S. Singhal. Web2K: Bringing QoS to web servers. (HPL-2000-61), 2000. `http://www.hpl.hp.com/techreports/2000/HPL-2000-61.pdf`.

[23] G. Starnberger, L. Froihofer, and K. M. Goeschka. Using smart cards for tamper-proof timestamps on untrusted clients. In *Availability, Reliability and Security, 2010. ARES '10. International Conference on*, Kraków, Feb. 2010.

[24] Transaction Processing Performance Council. TPC Benchmark$^{\mathrm{TM}}$W (Web Commerce), 2002. `http://www.tpc.org/tpcw/`.

[25] T. Voigt and P. Gunningberg. Adaptive resource-based web server admission control. In *ISCC*, pages 219–224. IEEE Computer Society, 2002.

[26] T. Voigt and P. Gunningberg. Handling multiple bottlenecks in web servers using adaptive inbound controls. In G. Carle and M. Zitterbart, editors, *Protocols for High-Speed Networks*, volume 2334 of *Lecture Notes in Computer Science*, pages 50–68. Springer, 2002.

[27] C. K. Yeo, B. S. Lee, and M. H. Er. A survey of application level multicast techniques. *Computer Communications*, 27(15):1547–1568, 2004.

[28] J. G. Ziegler and N. B. Nichols. Optimum settings for automatic controllers. *Transactions of the ASME*, 64:759–768, 1942.

[29] H. Ziyuan, Z. Lanlan, and F. Minrui. Robust auto tune smith predictor controller design for plant with large delay. In K. Li, M. Fei, G. W. Irwin, and S. Ma, editors, *LSMS (1)*, volume 4688 of *Lecture Notes in Computer Science*, pages 666–678. Springer, 2007.