Budget-based Control for Interactive Services with Adaptive Execution

Yuxiong He[1] Zihao Ye[2] Qiang Fu[2] Sameh Elnikety[1]
[1]Microsoft Research Redmond
Redmond, WA, USA
[2]Microsoft Research Asia
Peking, China

ABSTRACT

We study the problem of managing a class of interactive services to meet a response time target while achieving high service quality. We focus here on interactive services that support adaptive execution, such as web search engines and finance servers. With adaptive execution, when a request receives more processing time, its result improves, posing new challenges and opportunities for resource management.

We propose a new budget-based control model for interactive services with adaptive execution. The budget represents the amount of resources assigned to all pending requests. The budget-based control model consists of two components: (1) a hybrid control mechanism, which combines adaptive and integral controllers and controls the budget in order to meet the response time target with small steady-state error, fast settling time and little runtime overhead, and (2) an optimization procedure, which takes advantage of adaptive execution to maximize the total response quality of all pending requests under a given budget.

We implement and evaluate the budget-based control model experimentally in Microsoft Bing, a commercial web search engine. The experimental results show that it achieves more accurate control of mean response time and higher response quality than traditional static and dynamic admission control techniques that control the queue length. We also apply the model to a finance server that estimates option prices, and conduct a simulation study. The simulation results show large benefits for budget-based control. For example, under the same response time and quality requirements, the budget-based model accommodates double the system throughput compared to a traditional queue-based control model.

1. INTRODUCTION

Interactive services such as web search, web content servers, finance servers and online gaming serve millions of customers using thousands of servers. The SLA of these services often specifies stringent response time requirements. The most common metrics include mean and high-percentile response time. Long response times are not acceptable because they cause user dissatisfaction and revenue loss [21]. In addition to response time requirements, interactive services need to achieve high result quality. For example, a web search engine should return the most relevant web pages to user queries; a finance server needs to estimate price of the finance derivatives with small estimation error. Designing interactive services to meet their response time and quality requirement is an important and challenging problem.

To meet response time requirements, a common approach is to limit the length of the incoming request queue: when the queue is full, new requests are dropped upon arrival. Intuitively, the bounded queue length provides bounded waiting time that potentially leads to a desired response time target. However, such a static admission-control approach [22] leads to several undesirable situations which result in the failure to meet a response time target or in degraded quality. If the queue limit is too small, available resources become underutilized and response quality is degraded. A large queue limit, on the other hand, may result in violating response time requirements. Determining the appropriate static queue length limit is challenging in data center environments since systems change: the incoming load fluctuates over time; software updates and hardware upgrades affect request service demand; and service SLAs change to reflect the evolving business requirements. Moreover, while a well-designed interactive service should not be persistently overloaded, transient periods of overload are often inevitable. The load increase at the server that leads to a transient period of overload is often difficult to predict [3]. These factors suggest the need for self-managed systems that can adapt to the changes and meet response time and quality requirements.

An intuitive way to offer a response time guarantee in a changing environment is to apply feedback control. Feedback control has been widely used to achieve performance guarantees in many applications [5, 6]. Prior work [8, 9, 20, 23, 24] adjusts the queue length limit dynamically (or equivalently the buffer size or request drop rate) according to the feedback on response time: when the measured response time is higher than the target response time, the queue length limit is decreased; when response time is lower than that the target, the limit is increased. This type of dynamic admission control is effective for the classic "binary" request model: The server either processes the request returning a complete response or drops it with a null response. We call this model *binary* because the scheduler has a binary decision to make: either accept or reject the request.

In contrast to the binary request model, many online services support adaptive execution: a request may have several partial results with different qualities depending on the amount of received processing time. The request has a quality function that maps the received processing time to a corresponding response quality; the response quality often improves with more processing time. Many important applications follow the adaptive model, including the following: (1) Web search: A web search engine receives requests from clients and returns the matching webpages within a short deadline. For a web query, there are multiple acceptable answers, and more processing time allows the search engine to match and rank more webpages online, providing progressively better responses. (2) Finance servers: Traders interactively submit requests to estimate the price of financial derivatives. A finance server executes a computation, such as Monte Carlo based pricing algorithms, and the server can trade-off more processing time for smaller error between the estimated price and the real price.

Applying feedback control to adaptive interactive services is challenging. In addition to accepting or rejecting a request, the scheduler can execute requests partially: it needs to assign some processing time to each request based for example on request quality function or on system load. The goal is to meet the response time target and to provide high response quality. Existing approaches for the binary request model are not suitable here: they do not consider partial execution of requests, and therefore some queries are fully served while the others are rejected. Although traditional approaches may still meet the response time target; they result, however, in large degradation in the service quality, and they bring inconsistent user experience as some requests receive no service.

We exploit the problem of scheduling interactive requests with adaptive execution to meet the response time target while achieving high response quality. We introduce a budget-based control optimization model where the budget is defined as the total execution time (or amount of resources) for all pending requests. The model consists of two components: a feedback control mechanism to adapt the budget so the system can meet its desired response time, and an optimization procedure that schedules requests within the provided budget. The optimization procedure assigns the execution time of individual requests based on their service demands and quality profiles to improve their total response quality.

For interactive services, the control mechanism should adapt to the changing workload quickly and accurately while being simple enough with little runtime overhead. We develop a hybrid control mechanism combining adaptive and integral control to achieve this objective. Given the complexity of real systems, it is often hard to quantify the transfer function between system input and output in a changing environment. Therefore, we adopt a linear quadratic (LQ) adaptive control mechanism, which performs recursive linear regression to capture system behavior and uses an LQ optimal controller to compute the control output. However, frequent regression evaluation is too expensive for interactive services since each request takes on average only 20 ms of execution time. In addition, the pure LQ adaptive control cannot eliminate steady-state error. We, therefore, perform regression evaluation for adaptive control only at coarse-grain intervals, and within a coarse-grain interval, we apply integral control to adjust the control output, reducing both runtime overhead and steady-state error.

The budget-based control model applies the optimization procedure to exploit adaptive execution. With adaptive execution, a request can be partially processed, and a quality function maps the received processing time to the response quality. We develop two optimization procedures for two types of scheduling scenarios: (1) clairvoyant scheduling: the scheduler knows request service demand at their arrival, and (2) nonclairvoyant scheduling: the scheduler does not know request service demand until the request completes. The optimization procedure assigns processing time to pending requests using the budget determined by the control mechanism to improve the response quality.

We assess the benefits of the budget-based control model through system implementation and experimental evaluation as well as through a simulation study.

We show that the budget-based control model is feasible in practice, and we implement and evaluate it experimentally in Microsoft Bing, a large commercial web search engine. We employ hybrid control combining adaptive and integral controllers to meet the target mean response time with small steady-state error, fast settling time and little runtime overhead. The hybrid controller adjusts the budget. We measure the quality profile of Bing search requests and exploit the concavity of the quality profile to design an optimization procedure to schedule requests. The experimental results show significant benefits of using budget-based control over controlling the queue limit statically or dynamically. In particular, budget-based control meets the desired response time target and achieves high result quality. Moreover, since many commercial applications specify SLA using high percentile response time, we also apply the budget-based model to control the 90-percentile response time for web search and show its effectiveness.

We also evaluate the budget-based control model with a different optimization procedure in a finance server. We build a simulator that models a finance server using Monte Carlo methods to evaluate option prices. Here each request is a task to estimate the price of a financial option, and the response quality is measured by the price estimation error: the smaller the estimation error, the better the result quality. Our results show that, to meet the same mean response time target, the budget-based control model reduces the estimation error and improves response quality compared to the queue-based control model. In particular, to achieve the same target quality, the budget-based control model doubles the system throughput while satisfying the target mean response time.

The contributions of this paper are as follows: (1) We propose the budget-based control model for interactive services with adaptive execution. (2) We introduce a hybrid control mechanism suitable for interactive services, which combines adaptive and integral control to meet response time requirements. We also introduce two optimization procedures to improve response quality for clairvoyant and nonclairvoyant scheduling environments. (3) We show how to implement budget-based control in modern servers as used in Bing. (4) We evaluate the benefits of budget-based control experimentally, using real server software and workload in Bing. (5) We conduct a simulation study to evaluate budget-based control in a finance server.

The paper is organized as follows. Section 2 discusses adaptive execution and presents the measured quality profile from Bing. Section 3 describes the budget-based control model, and its two components: the control mechanism and the optimization procedure. Section 4 describes the implementation and experimental evaluation results in Bing. Section 5 describes the simulation results in a finance server. Section 6 discusses related work and Section 7 shows our conclusions.

2. Adaptive Execution

Adaptive execution is the flexibility of trading more resources for better results. Examples include estimation computations in which better estimates are obtained with more processing. For example in a finance server, Monte Carlo methods compute option pricing to find better answers with more processing.

With adaptive execution, the relationship between the quality of the result and the used amount of computational resources is quantified by request quality function. A quality function $f: R \rightarrow R$ maps the request completion ratio (processing time / service demand) to a quality value gained by executing the request. Quality functions of different applications can have different shapes. We observe that the quality functions are usually monotonically non-decreasing: result quality stays the same or improves with more processing. Moreover, many best-effort

applications exhibit concave quality profiles due to the effects of diminishing returns.

To demonstrate the quality profile for a real application, we measure the response quality profile in Bing using $200K$ queries from a production trace and present it in Figure 1. Each request is a web search query for a set of keywords. The search engine scans its inverted index looking for webpages that match the requested keywords and ranks the matching webpages. The more time the server spends on matching and ranking the webpages from the inverted index, the better the results. The response includes a set of links to the top webpages matching the keywords specified by the user query. The response quality compares the set of webpages returned in the test to a golden set of base results (as explained in Section 4). The x-axis of Figure 1 is request completion ratio; the y-axis is the average response quality. The figure demonstrates that the quality profile of Bing search is monotonically increasing and concave: the concavity comes from the effect of diminishing returns. The inverted index lists important (e.g., popular) webpages first; therefore webpages matched earlier are more likely to rank higher and contribute more to total response quality.

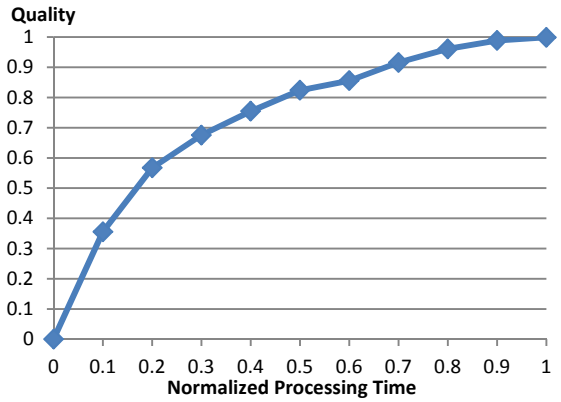

Figure 1. Measured quality profile in Bing search.

A scheduler can exploit adaptive execution and request quality profile to improve the response quality. Without adaptive execution, a scheduler can either execute a request fully or reject it. In contrast, adaptive execution opens the possibility of partial processing in which some requests can be processed half-way returning an approximated result, which is a favorable trade-off in many situations.

3. Budget-based Control Model

This section presents the budget-based model for interactive services with adaptive execution. The budget is defined as the total processing time (or total amount of resources) we plan to allocate to all pending requests. The budget-based control model consists of two components: (1) the control mechanism, which applies feedback control to adjust the budget to meet the response time target, and (2) the optimization procedure, which is a scheduling algorithm using adaptive execution to maximize total response quality at a given target. The budget-based model divides the two main goals of the system, meeting response time target and achieving high quality, into the control mechanism and optimization procedure respectively, so each component has a clear design goal and responsibility. This section elaborates the design and features of the control mechanism and optimization procedure.

3.1 Control Mechanism

The control mechanism takes the observed response time as feedback and determines the budget value in order to meet the response time target. There are several approaches to design such a controller. We present a hybrid controller, combining adaptive and integral controller, which offers accurate and light-weight control of interactive services to meet their response time requirements. Our experimental results show that such hybrid control combines the benefits of adaptive and integral control: It meets the response time target with small steady-state error, fast settling time and little runtime overhead, outperforming the adaptive controller alone and integral controller alone. This section first gives a brief description of the integral and adaptive controllers then it introduces the hybrid controller.

3.1.1 Integral Control

Integral control is a simple and well-known control mechanism, adjusting the value of control variable based on the difference between the observed output and the reference output. The control function is expressed as follows:

$$u(k) = u(k-1) + K_I e(k)$$

Here k represents time steps; $u(k)$ is the output of the integral controller (which is also the control variable of the system) at time step k. The tracking error $e(k)$ is the difference between the observed system output and the reference output, i.e., $e(k) = y_{\text{ref}}(k) - y(k-1)$. The controller parameter K_I defines the ratio of control change to the control error. In our system, the control variable u is the budget, and the output y is the metric such as mean response time, or the 90-percentile response time.

Integral control has two main advantages. First, it has zero steady-state error, which allows systems to meet their desired SLA. Second, it is computationally efficient, which is an important property in interactive systems, The integral controller incurs almost negligible system overhead allowing recalculating the budget with each request arrival or departure.

Integral control has its limitation: Its response is relatively slow. For example, when the workload has a big change, it will track the change slowly, producing a large deviation initially. Integral control is usually combined with another control mechanism to overcome this limitation.

3.1.2 Adaptive Control

We consider a linear quadratic adaptive controller [15] with two parts: a model estimator and a linear quadratic optimal controller.

The Model Estimator. The model estimator uses the prior behavior of the system to predict the current system model. More precisely, it uses a number of prior control input and output values to predict their relationship using a Recursive Least Square (RLS) model estimation. In our problem, adaptive control predicts the relationship between the budget and response time using a linear function. The order of the regression model (i.e., the number of prior data points used for predicting the system model) indicates a tradeoff between the estimation precision and computational overhead. With a larger order, the estimation is generally more precise, however, the regression computation incurs a higher computation overhead.

Linear Quadratic (LQ) Optimal Controller. The primary control objective is to make the system output track the desired SLA with small error. It is also desirable to avoid large changes to the control variables. These two goals are achieved by minimizing the quadratic cost function F defined as follows for linear quadratic optimal control:

$$F = |W(y_{est}(k+1) - y_{ref}(k+1))|^2 + |Q\big(u(k) - u(k-1)\big)|^2$$

Here $y_{est}(k+1)$ denotes the estimated system output from the estimator, $y_{ref}(k+1)$ denotes the reference output for the step $k+1$, and $u(k)$ and $u(k-1)$ are the control variables (or system inputs) at time step *k* and step *k-1*, respectively. Moreover, W is a positive weighting parameter on the tracking errors, and Q is a positive weighting parameter to penalize large changes in the control variable. The relative magnitude of W and Q trades off between tracking accuracy and smaller changes in the control variable. The value of the control variable $u(k)$ that minimizes F can be obtained by setting the derivative $\partial F / \partial u(k) = 0$.

Next we turn to the advantages of adaptive control. Interactive services are complex systems: they use heterogeneous hardware crossing different data centers and generations of software and hardware components; the workload fluctuates over time; software updates and hardware upgrades affects the request service demand. More fundamentally, when an interactive service supports adaptive execution, requests can be partially processed. Therefore, it is hard to quantify the transfer function between the control variable (in our case, it is the budget) and the response time. This makes adaptive controller a good choice: it captures the system behavior by modeling the correlation between budget and response time using the recent data points. The RLS-based model estimator identifies the changes in the system behavior, and the controller adjusts the control output and adapts the system correspondingly.

There are limitations of adaptive control. (1) It is computationally expensive for interactive services. Given a regression order N, i.e., the estimator uses the latest N data points (i.e., control input, control output), it requires $O(N^3)$ arithmetic operations at each time step to compute the prediction model online. Our measurements indicate that it takes about 2.2 ms on our server when $N = 10$. The average service demand of a request is around 20 ms. If adaptive control is applied on every request, it introduces fairly large overhead to the system. (2) The LQ adaptive controller, similar to a proportional controller, incurs steady-state errors. More specifically, if we set Q to 0, the LQ has the form of a proportional controller which can't guarantee the small steady-state errors.

3.1.3 Hybrid Control

We develop a hybrid controller to combine the integral and adaptive control: it runs adaptive control periodically in a coarse-grain time interval to learn the system model and adjust the control output, the budget. Within the fine-grain interval, the hybrid controller uses integral control for the execution of each request to perform fine-grain adjustment of the control output. This combination reduces the computation overhead and the steady-state error. The block diagram of the hybrid controller is shown in Figure 2.

The hybrid controller uses both integral and adaptive control. If we use integral control only, we may have slow response for tracking changes in workload or system environment. Adding adaptive control allows learning the system model according to the recent system behavior. We cannot, however, afford the runtime overhead of using adaptive control for each request and there will be a steady-state error. Using integral control complements adaptive control, since integral control runtime overhead is almost negligible and it eliminates the steady-state error to meet the response time target. The adaptive part of the hybrid controller helps to adapt the system to large changes.

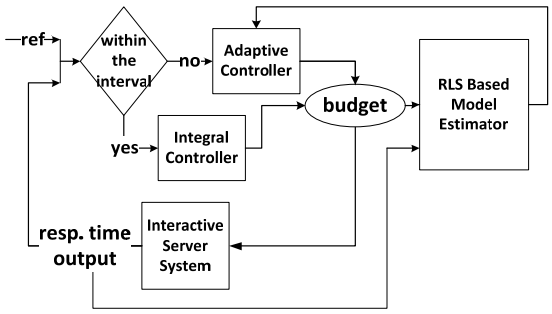

Figure 2: Hybrid Control Block Diagram.

Comparing the hybrid controller with a classic PI controller, the parameter of the P controller is fixed and difficult to tune for different loads and configurations, while the adaptive control of the hybrid controller adapts the system on different hardware configurations and workload regions automatically.

3.2 Optimization Procedure

The optimization procedure is a scheduling algorithm that takes a resource budget and a set of pending requests as inputs, and assigns a portion of the budget to each request with the objective of maximizing total response quality. The design of the optimization procedure depends on the request quality profile, service demand, and other application specific constraints. We do not intend to enumerate all optimization procedures to cover all scenarios. Rather, we focus on concave quality profiles as they are popular in practice due to the iterative nature of many best-effort applications and the effect of diminishing return. We introduce two optimization procedures for clairvoyant and nonclairvoyant scheduling environment respectively. This section presents how these two optimization procedures work and they are evaluated in Section 4 and 5. For other quality profiles and application specific constraints, one can develop and employ a tailored optimization procedure.

3.2.1 Optimization for Clairvoyant Scheduling

This section describes an optimization procedure with known request service demand and concave quality profile. An example is the finance server that uses Monte Carlo methods to evaluate option prices (Section 5). This optimization procedure maximizes the response quality of the pending requests under a given budget.

Figure 3 presents the optimization procedure. The budget is defined as the amount of processing time available for all ready requests. The optimization procedure decides the assigned processing time of each request in the ready queue by solving the optimization problem defined at MaxQuality. MaxQuality maximizes the total quality of requests based on the budget, request demand and quality profile. Since all constraints in the MaxQuality are linear and its objective is to maximize the summation of concave functions, MaxQuality is a convex optimization problem and can be solved using convex solvers such as CVS [1]. MaxQuality produces a solution that maximizes the total quality of all pending requests with a given budget.

Without compromising the total response quality, we can further reduce response time by applying MinMRT after MaxQuality. MinMRT sorts the requests in the ascending order of the assigned processing time. It is well known that given a batch of jobs, running the shortest job first produces a schedule with the smallest mean response time [17]. Here by performing MinMRT after MaxQuality, the scheduler further minimizes the response time in the set of solutions that maximize the total quality. This also

benefits the response quality: given the same budget, MinMRT reduces the mean response time; thus given the same mean response time target, MinMRT relaxes the budget and leads to higher quality.

Optimization Model

Inputs:

$J = \{J_i | i = 1, \dots, n\}$: set of n ready jobs

w_i: service demand of job J_i

f_i: quality function of job J_i

B: budget (total processing time for all pending jobs)

Variables:

p_i: assigned processing time of J_i.

MaxQuality:

Maximize $\sum_{i=1}^{n} f_i(p_i/w_i)$

Subject to:

Resource availability constraint: $\sum_{i=1}^{n} p_i \leq B$

Processing time constraint: $p_i \leq w_i$

MinMRT:

Sort jobs in J in ascending order of p_i

Figure 3: Optimization procedure for clairvoyant scheduling.

The optimization procedure performs local optimization on the set of pending requests and the available budget. When there is new request arrival or budget change, we can repeat the optimization procedure with the updated request information.

3.2.2 Optimization for Nonclairvoyant Scheduling

This section describes an optimization procedure assuming requests have unknown request service demands and have concave quality profile. An example application is the index server at Bing (Section 4). Besides meeting response time and quality requirement, Bing server has two additional requirements: (1) It is often hard to know the exact quality profile of each request. The scheduler uses an expected quality profile (as shown in Figure 1) for all requests. (2) Context switching is expensive because of cache warm-up; it may take a few hundred microseconds to more than a millisecond [2]. Since the mean service demand of Bing requests is only about 20 ms, the scheduler should execute each request only once, rather than suspending the request and resuming it later.

Figure 4 shows the pseudo-code of the optimization procedure EqResv. The budget is defined as the amount of processing time available for all pending requests. The input does not include request service demand because it is unknown at request arrival. EqResv is a heuristic algorithm to improve total response quality of ready requests under a given budget.

EqResv processes requests in the FIFO order and it decides the assigned processing time of the first job in the FIFO queue based on the load and the budget. In order to improve total response quality, when requests are competing for resources, a scheduler prefers running the part of requests with higher quality gain. Given a concave quality profile, the early portion of processing request has higher gain than its later portion. Therefore, the key idea of EqResv is to prevent jobs at the beginning of the queue from consuming the entire budget and starving later requests so each request has a fair opportunity to be processed (at least for its early portion). To achieve this goal, EqResv applies two techniques. (1) Equi-Partitioning (EQ): When the system is heavily loaded, EqResv performs EQ to reserve a fair share of processing time for waiting requests (in Line 2). With a concave quality function, giving each job the same amount of processing time maximizes the overall quality. (2) Reservation (RESV): In a lightly loaded case, EqResv performs RESV to reserve the expected service demand for the queuing requests and allocates the remaining time to the current running job (in Line 3)[1]. RESV gives the long requests a chance to finish if they will not impact short ones.

EqResv (jobs[] *queue*, double $\bar{w}$, double B)

Inputs:

queue: list of ready requests in FIFO order;

$\bar{w}$: expected (or mean) service demand of requests;

B: budget (total processing time for ready jobs)

Pseudo code:

1: $qLen = queue.size()$ //queue length

2: $EQ = B / qLen$ // Equi-partitioning

3: $RESV = B - (qLen - 1) \times \bar{w}$ // Reservation

4: //assign processing time for the first job at ready queue

5: $queue[0].p = max(EQ, RESV)$

Figure 4: Optimization procedure for nonclairvoyant scheduling.

EqResv does not need a load threshold to decide if it should use the result from EQ or RESV. During light load, we want to estimate the processing time using RESV, and its processing time is larger than the one produced by EQ. During heavy load, we want to use EQ, and its processing time is larger than the one produced by RESV. Therefore, selecting the larger between these two gives the assigned processing time (in Line 5).

4. Implementation and Evaluation in Bing

This section presents the implementation and evaluation of the budget-based control model in Microsoft Bing web search engine.

4.1 Application Overview

Bing is a large commercial web search engine from Microsoft. We focus on the index serving part (interactive processing), which serves user queries online to return the best matching webpages. Notably, the index serving part is different from the web crawler and index builder (batch processing) which processes crawled webpages to generate the inverted index offline.

The index serving system of Bing accepts user queries, and it forwards the queries to index servers when the query's results do not exist in the cache. Each index server manages a small portion of the inverted index and therefore becomes responsible for a set of web pages. The index server searches its inverted index for all

[1] Service demand of individual request is unknown but expected or mean service demand of requests can be obtained through offline measurement or online approximation.

webpages that match the query, ranks these webpages, and returns the top N webpages that match the query. The index server supports adaptive execution: the result quality improves with the increased number of webpages examined and ranked. Moreover, the response quality profile is concave as shown in Figure 1. In Bing, we want to limit the mean response time of the index server as part of the web search SLA requirements for important commercial reasons. These factors make the index server a good candidate to apply the budget-based control model.

4.2 Implementation

The web index is partitioned among many index servers. We use our approach to control each index server so that each individual server can satisfy the target mean response time while returning high quality results. The original index server works as follows. Newly arrived requests join the waiting queue. The waiting queue has a length limit: when the queue is full, new requests are dropped. There are a number of worker threads and each worker thread processes one request at a time. The number of workers is equal to the number of cores in the system. When a worker thread completes a request, it gets a new query from the head of the waiting queue and starts to process it. To process a query, the worker searches the inverted index and obtains a list of matching webpages to the search keywords. It then ranks the matching webpages in a loop, which we call index ranking loop. This loop is the most time consuming part of the query processing. After ranking all matched webpages, the worker returns the top N matched results and completes the query.

Our implementation at index server includes three parts to apply budget-based control.

(1) We enable adaptive execution of requests using early termination. We add a termination condition in the ranking loop, so that when a request uses up its assigned processing time, the request is terminated early.

(2) We add the optimization procedure from Figure 4 to dynamically assign processing time to requests based on the budget. The optimization procedure is extended to multicore servers by changing the queue length value (qLen) to reflect the expected queue length for each core.

(3) We implement the hybrid controller consisting of the model estimator and linear quadratic optimal controller to adjust the budget based on mean response time. The mean response time is computed online as a moving average $MRT = (1-\alpha) \times MRT + \alpha \times r_i$, where r_i is the response time of the last processed request i and α is a constant multiplier. We use $\alpha = 0.05$ in our implementation.

To compare the budget-based control to the traditional dynamic approaches, we also implement queue-based control and integrate it in the index server. It applies the same hybrid controller in Section 3 to adjust the queue length limit based on the mean response time. When the queue is full, the newly arrived requests are dropped. When the queue length limit decreases due to control decision, the overflowed requests are also dropped.

In the remainder of this section, we compare these three implementations of the index server:

- OriginalIS: original implementation of index server
- BudgetIS: index server using budget-based control
- QueueIS: index server using queue-based control.

4.3 Experimental Setup

Performance Metrics: The primary goal to control an interactive system is to meet the response time target and to achieve high response quality. The index server has an SLA requirement on the average response time and response quality. The request response time is the duration between when request arrives to the index server and when the response is sent back; the server sends responses to all requests including the dropped ones. Our experiments use 35 ms as our target mean response time, and we control the server to make the mean response time at or below the target. We also tried several other mean response time targets and the results are similar.

To compute the quality of a response of a web search query, we compare returned webpages in the response to the webpages in the base results of the query when it is processed completely. We use proportional quality, which gives each of the top N webpages the same weight. For example, when N=10 and there are 8 matches between the response and the base results, the quality is 8/10. Proportional quality is one way to measure the response quality. We also used other quality metrics, such as assigning higher weights to higher ranking webpages; the experimental results are similar we, therefore, present the proportional quality only.

Other important measures include classic metrics for evaluating controllers, such as settling time and steady-state error. Settling time is the time from the change in the workload to when the measured output is sufficiently close to its new steady-state value. Shorter settling time is desired. Moreover, the control mechanism should be computationally efficient without incurring high overhead.

Workload and Hardware: Our evaluation includes an index server that answers queries and a client that replays queries from a trace file. We use a query trace with 200,000 actual user queries from production to drive the experiments. We run the system by issuing queries following a Poisson distribution in an open-loop system. We vary system load by changing the arrival rate expressed as QPS (queries per second). The index server searches its local index and returns the top 10 matching results to the client. The index server for our evaluation has a six core Intel 64-bit Xeon processor (3.33 GHz) and 24 GB main memory.

Controller Configurations: The hybrid controller uses adaptive control to adjust the budget at every 10 queries and applies integral control at every step with $K_I = 1$. The order of the RLS-based model estimator is 10. The weight parameters of the linear quadratic controller have values Q=0.5 and W=0.5.

Experiments: We conduct the following four sets of experiments and present their results in the remaider of this section.

(1) Comparing budget control to static queue: we compare the budget-based control model (BudgetIS) to the original index server that has a static limit on the queue length (OriginalIS).

(2) Control variables: we compare BudgetIS that uses budget as control variable to QueueIS that uses queue length.

(3) Control mechanisms: we evaluate the impact of integral, adaptive, and hybrid control mechanisms.

(4) Controlling high-percentile response time: we apply budget-based control to meet a 90-percentile response time target.

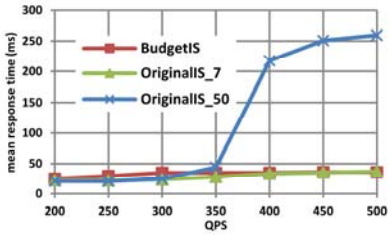

Figure 5: Mean response time for BudgetIS and OriginalIS.

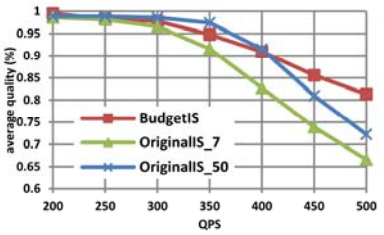

Figure 6: Average quality for BudgetIS and OriginalIS.

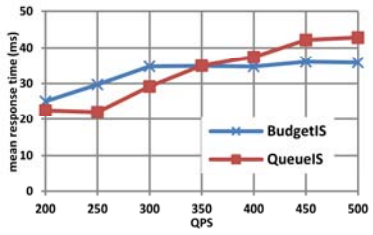

Figure 7: Mean response time for BudgetIS and QueueIS.

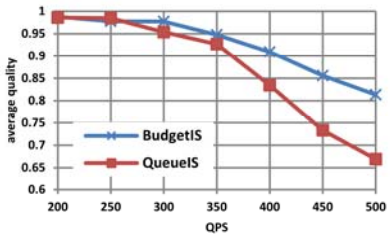

Figure 8: Average quality for BudgetIS and QueueIS.

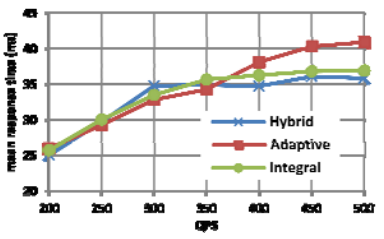

Figure 9: Mean resp. time for different control mechanisms

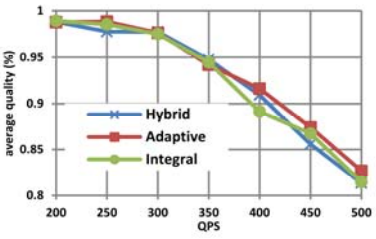

Figure 10: Average quality for different control mechanisms.

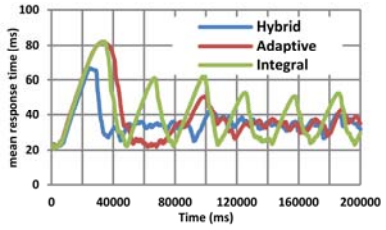

Figure 11: Transient behavior.

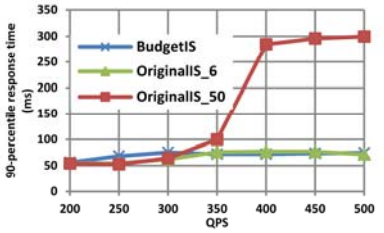

Figure 12: P90 response time for BudgetIS and OriginalIS.

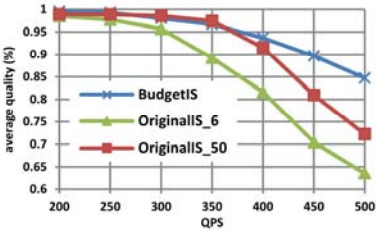

Figure 13: Average quality for BudgetIS and OriginalIS.

4.4 Comparing Budget Control to Static Queue

This experiment compares BudgetIS to the static approach of having a fixed queue length. We use two fixed queue length values 7 and 50 (corresponding to OriginalIS_7 and OriginalIS_50 respectively) to demonstrate the effect of different queue lengths. Figures 5 and 6 show the mean response time and quality results for the three implementations: their x-axis represents the load expressed as the request arrival rate, varying from 200 to 500 QPS, which covers the operational range of the workload. The y-axis is the mean response time and average response quality, respectively.

The results show that for static approach, different queue lengths represent a tradeoff between mean response time and average response quality. With shorter queue length, the mean response time decreases and the quality degrades too. OriginalIS_7 meets the response time target, but its quality degrades even at light and moderate load; OriginalIS_50 obtains higher quality but its mean response time at high load is significantly higher than the target. It indicates that there is no single queue length value that meets both the response time and quality requirement.

Figure 5 shows that BudgetIS successfully bounds the request mean response time to the 35 ms target with tracking error less than 1 ms. Moreover, Figure 6 shows that BudgetIS also improves response quality at high load. In particular, the response quality of OriginalIS_7 and OriginalIS_50 drops more sharply at high load and OriginalIS_50 exceeds the response time target, while BudgetIS offers higher quality. BudgetIS achieves this by assigning request processing time dynamically exploiting request quality profile to improve the total response quality. For example at 450 QPS, BudgetIS terminates 32% queries early with partial results and no queries are dropped, while OriginalIS_7 drops 25% queries and these queries have quality 0.

This experiment demonstrates that (1) Using a fixed queue length cannot meet response time requirements with high response quality, and (2) The budget-based model accurately controls the mean response time to match the SLA target and uses partial evaluation to improve the request response quality.

4.5 Control Variables: Budget vs. Queue Length

This experiment compares QueueIS, which controls the length of the waiting job queue, to BudgetIS, which controls the budget. Figures 7 and 8 show the mean response time and quality comparison for the two approaches. Both approaches can bound request mean response time at high load, however, QueueIS incurs bigger errors tracking the response time target and has worse quality at high load.

QueueIS incurs bigger tracking error than BudgetIS for two factors. (1) QueueIS uses queue length as control input, which is an integer value with the smallest change of incrementing or decrementing by one; the discrete values of the control input may not be able to meet the control target precisely. (2) In BudgetIS, changes in the budget value are immediately reflected on the queries' processing times, since the system assigns query processing time according to the budget. However, in QueueIS, changes of the queue length take effect only after a period of time, since queue length won't affect response time of the queries before the queue becomes full. Such a delay between control input and output can also cause reduced control accuracy.

QueueIS produces lower quality than BudgetIS at high load because QueueIS drops queries to meet the response time target while BudgetIS processes queries partially. Given a concave quality profile, partially executing queries with similar processing time achieves higher average quality than executing some queries fully while dropping the others.

4.6 Comparing Control Mechanisms

This section shows that the hybrid control mechanism which combines integral and adaptive control outperforms either integral control alone or adaptive control alone. Hybrid control offers small steady-state errors, small settling time and is computationally efficient. Adaptive and integral control offer a subset of these properties rather than all of them.

In this experiment, all the evaluated systems use the budget as the control variable with the same optimization procedure but different control mechanisms.

Figure 9 and 10 show mean response time and average quality for the different control mechanisms with the system load. We discuss each of them below.

Adaptive controller. The adaptive controller exceeds the mean response time target of 35 ms at high load. It uses the RLS model estimator to predict system behavior and its control law is close to proportional control: its accuracy is sensitive to workload variation and the control law cannot eliminate steady-state errors. Moreover, running model estimation of adaptive control before executing every request introduces a considerable amount of computational overhead (about 2.2 ms of overhead for every query with average service demand of 20 ms). This not only increases the mean response time of requests but also becomes the noise factor that the control law cannot remove from its steady-state error. Therefore, using adaptive controller alone cannot bound mean response time effectively.

Integral controller. The integral controller controls the mean response time effectively with tracking errors less than 2.5 ms. However, it has a long settling time. Figure 11 compares the transient state behavior of integral controllers to the hybrid and adaptive controllers. In this experiment, we first launch queries at 200 QPS; then we double the load to 400 QPS. The figure shows that hybrid controller has the shortest settling time. As for the integral controller, it has slower responsiveness to the workload change, with large settling time. Due to its slow responsiveness, the integral controller does not meet all the desired properties.

Hybrid controller. The hybrid controller has the best characteristics among the three control mechanisms: the smallest steady-state error, highest response quality, and the shortest settling time. It uses the adaptive controller in a coarse-grain manner to detect large changes and responds quickly; it uses the integral controller in a fine-grain manner to reduce steady-state error and reduce computation overhead. The hybrid controller combines the advantage of adaptive and integral controller.

4.7 Controlling High-Percentile Response Time

High-percentile response time is another important and common SLA requirement for interactive services. This section shows that the budget-based model meets the high percentile response time target. This experiment is conducted on Bing index server with a 90-percentile response target of 75 ms, i.e., 90% requests must have response time of 75 ms or less. We use the last 1000 queries' 90-percentile value as the current observed value in a moving window of recent requests and we adjust the budget based on the difference between the observed and the target 90-percentile response time.

Figure 12 and 13 show the 90-percentile response time and average response quality for BudgetIS, OriginalIS_6 and OriginalIS_50. The results are similar as in Section 4.4. BudgetIS effectively meets the 90-percentile response target while OriginalIS_50 incurs very high response time at heavy load and OriginalIS_6 suffers from quality loss at light and moderate load. The quality of both versions of OriginalIS is lower than BudgetIS at heavy load due to request dropping. Again, the benefits of BudgetIS come from adopting partial results and exploiting the concave quality profile. This experiment demonstrates that the budget-based model is not limited to controlling mean response time; it can be extended to meet other SLA for adaptive interactive services.

5. Finance Server

Section 4 evaluates the budget-based control model for nonclairvoyant scheduling where request service demand is unknown. This section evaluates it for clairvoyant scheduling. We build a simulator to model a finance server where request service demand is known. We show that budget-based control model outperforms the queue-based model: under the same load, the budget-based control model produces higher response quality and under the same quality requirement, it achieves higher throughput.

5.1 Application Overview

Banks and fund management companies evaluate thousands of financial derivatives every day. Traders and analysts submit requests to value the derivatives, and they make trading decisions online based on the returned results. At the backend, there are many servers that perform quantitative analysis on various financial products. This section presents an option pricing server that uses Monte Carlo methods to price complex path-dependent options. Monte Carlo methods are widely used for analyzing complex derivatives that are difficult to value using other techniques such as Black-Scholes and lattice-based computations [16]. Monte Carlo methods are computationally intensive and rely on repeated random sampling to compute the results. Such a finance server is a good candidate for budget-based control: (1) tasks are time-bounded: traders often wait for no more than a few seconds to get the results and perform online trading and (2) tasks are adaptive: with more processing time, the price estimation error reduces and result quality improves.

5.2 Performance Metric and Quality Profile

The result quality is measured by a statistical metric called standard error of mean (SEM), which is the standard deviation of the sample mean to the population mean [4]. It indicates how well the sample mean estimates the population mean. The SEM value is calculated as the population standard deviation [2] divided by the square root of the sample size n, i.e., $SEM = \delta/\sqrt{n}$. The smaller SEM is, the closer the estimated price is to the real price.

Figure 14 shows the request error profile[3] with the normalized processing time, which is the ratio of request processing time to

2 The population standard deviation is often unknown in practice. As a conventional technique, we estimate SEM using the sample standard deviation divided by the square root of the sample size.

3 When total processing time of a request is 0, SEM value is undefined and can be arbitrarily large. To compute mean SEM of requests including the unprocessed ones, we set the unprocessed request with

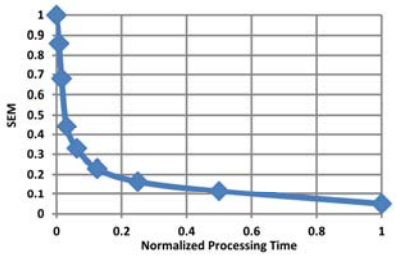

Figure 14: Error profile.

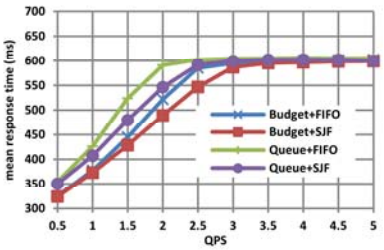

Figure 15: Mean resp. time comparison.

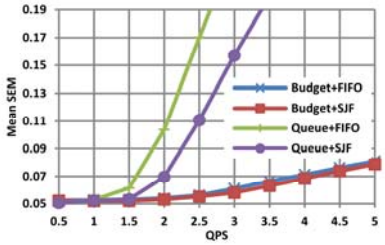

Figure 16: Mean SEM comparison.

its full service demand. Here we set the SEM target to 0.05: when a request's SEM reaches 0.05, we consider it fully evaluated. When the number of samples increases along with the processing time, SEM decreases, which indicates the increase of the result quality. Moreover, the error profile is convex; when we compute more samples, the additional reduction on error for adding a sample gets smaller. Here minimizing SEM with a convex error profile is equivalent to maximizing quality with a concave quality profile. Smaller SEM indicates better quality.

Request service demand is known because it depends on two input values of the option, namely (1) the total duration and (2) period value, and the target SEM, which are all known at request arrival.

5.3 Experimental Setup

In the simulation study, requests arrive following a Poisson process, and their service demands follow an exponential distribution with an average of 300 ms. The desired mean response time is 600 ms. We implement and compare two control models: budget-based and queue-based control. Moreover, since request service demand is known at request arrival, we also apply the shortest job first (SJF) technique [17] to reduce mean response time. So in total, we evaluate four schemes:

- Budget+FIFO: budget-based model with optimization procedure in Figure 3 with MaxQuality only (and without MinMRT). The requests are processed in FIFO order.
- Budget+SJF: budget-based model with optimization procedure in Figure 3 with both MaxQuality and MinMRT. The requests with smaller assigned processing time are processed earlier.
- Queue+FIFO: queue-based model serving requests in FIFO order and dropping a new request when the queue is full.
- Queue+SJF: queue-based model serving requests using SJF ordering and dropping the longest request when queue is full.

5.4 Performance Evaluation

Figure 15 and 16 show the mean response time and SEM of the four schemes with the varying load expressed as QPS or user requests per second. All schemes effectively bound mean response time at 600 ms or below, but budget-based schemes produce much smaller SEM and thus higher quality. For example, to keep $SEM \leq 0.1$, the maximum throughput which the queue-based approach sustains is less than 2.5 QPS while budget-based approach can sustain more than 5 QPS, which doubles the throughput. We now look into more details of the results.

From Figure 15, all schemes effectively bound the mean response time under 600 ms. SJF helps to reduce mean response time at moderate load: both Budget+SJF and Queue+SJF exhibit lower response time than their corresponding FIFO versions at load 1.5-2.5 QPS. At light load, SJF is similar to FIFO because most jobs don't wait and mean response time is close to mean service demand. At heavy load, again, SJF is similar to FIFO because response time is controlled around the 600 ms target value.

There are three observations from Figure 16. (1) Budget-based schemes show much lower error and thus higher quality than queue-based schemes because they use adaptive execution to achieve partial results and use quality profile to optimize the assigned processing time of requests for higher quality. (2) Queue+SJF achieves higher quality than Queue+FIFO because given the same queue length, SJF helps to reduce the mean response time; thus given the same mean response time, Queue-SJF may allow longer queue length than Queue-FIFO, which results in less dropped queries and higher quality. (3) The quality difference of Budget-SFJ and Budget-FIFO is very small. This seems to be inconsistent with observation (2), but it does not. Using optimization procedure MaxQuality at Figure 3, when requests have concave quality profile and they are competing for resources, long requests are likely to be cut to prevent them from starving the short requests. Therefore, at heavy load, requests tend to obtain nearly equal processing time such that using FIFO or SJF results in similar orderings, making little difference.

6. Related Work

Feedback control theory has been widely used to achieve performance guarantees in computer systems with many applications such as multimedia streaming, real-time computing, transaction processing, embedded systems, and many others [5, 6]. In this section we focus on server systems using feedback control to meet response time guarantees, and applications that use adaptive executions.

Controlling server systems with response time requirements. The prior works along this line focus on three scenarios.

(1) Control for relative response time. For example, Adbelzaher et al. [7] build a feedback control loop for an Apache web server that enforces desired relative response time among different service classes via connection scheduling and process reallocation.

(2) Control elastic resources. In these prior works [8, 9, 25, 26, 27], systems acquire and release resources in response to dynamic workload to meet response time target. There are various types of resources to adapt: For example, adding or removing a storage node [27], altering CPU allocation [25], changing processing speed through dynamic voltage and frequency scaling [26].

(3) Control to prevent overloading. While a well-designed system should not be persistently overloaded, transient periods of overload are often inevitable, since the load is external to the server system and requests arrive according to a stochastic process, leading to transient overload and underload periods at the server. Such transient periods are inevitable and difficult to predict [3]. Many prior works [20, 23, 24] apply feedback control

quality 1 (a small value in favor of queue-based model since it is likely to drop more requests.)

to cope with transient overload, deciding when to drop requests in order to meet response time target.

The above prior work [7, 8, 9, 20, 23, 24, 25, 26, 27] uses control theory to achieve response time guarantees, however, none of them consider adaptive execution of requests. Like many prior work [18, 19] on admission control, they either serve a request in full or reject a request completely. Our budget-based model is designed for applications with partial evaluation and it optimizes the scheduling based on request quality profiles.

Adaptive execution. Employing adaptive execution and approximate computations is an active area of research. Web content adaptation [10, 11] offers different versions of the content for the same request. Loop perforation [12] offers compiler and runtime support for adaptive execution and has been applied to audio and video codecs. Baek and Chilimbi [13] develop a general framework to support approximated computation of different applications to trade quality for lower energy.

These prior works [10, 11, 12, 13] offer important insights on how to adapt execution for different applications. They focus on adaptive execution mechanism that enables individual requests to produce partial results. They do not, however, consider server environments where multiple requests are competing for resources with response time and quality targets.

Control systems with content adaptation. The closest prior work to ours is controlling web servers that support content adaptation, which is a form of adaptive execution. Abdelzaher and Bhatti [14] propose to resolve the overloading problem of web servers by adapting web content to load conditions. To meet the desired server utilization, they control the ratio between the requests offering degraded content versus all the requests. Although this work uses adaptive execution to meet their control target, it has important differences from our work: they do not consider maximizing overall response quality for all requests as a goal, and they do not consider request quality profiles to improve the scheduling decision. We develop the budget-based model as a general approach for interactive services supporting adaptive execution. With an appropriate optimization procedure, it is applicable to web servers with content adaptation.

7. Conclusions

This paper presents the budget-based control model for interactive services with adaptive execution to meet a response time target while achieving high service quality. The budget-based model consists of two components: (1) a hybrid control mechanism that adapts the budget so as to meet the response time target accurately and quickly, and (2) an optimization procedure that improves the total response quality using adaptive execution. We assess the benefits of the budget-based control model through system implementation and experimental evaluation on a commercial search engine as well as through a simulation study of a finance server. Both the experimental and simulation results show that the budget-based model achieves more accurate control of mean response time with higher response quality than the traditional static and dynamic approaches that do not consider adaptive execution.

8. References

[1] J. Hiriart-Urruty and C. Lemar´echal. Convex Analysis and Minimization Algorithms, I and II. 305 and 306. 1993.

[2] C. Li, C. Ding, and K. Shen. Quantifying the cost of context switch. ECS, 2007.

[3] B. Schroeder and M. Harchol-Balter. Web servers under overload: How scheduling can help. ACM Trans. on Internet Tech. 2006.

[4] http://en.wikipedia.org/wiki/Standard_error.

[5] J. L. Hellerstein, Y. Diao, S. Parekh, and D. M. Tilbury. Feedback Control of Computing Systems. 2004.

[6] T. Abdelzaher, Y. Diao, J. L. Hellerstein, C. Lu, and X. Zhu. Introduction to control theory and its application to computing systems. Performance Modeling and Engineering. 2008.

[7] C. Lu, T.F. Abdelzaher, J. Stankovic, and S. Son. A feedback control approach for guaranteeing relative delays in web servers. RTAS, 2001.

[8] L. Sha, X. Liu, Y. Lu, and T. Abdelzaher. Queuing model based network server performance control. RTSS, 2002.

[9] X. Liu, R. Zheng, J. Heo, Q. Wang, and L. Sha. Timing performance control in web server systems utilizing server internal state information. ICAS/ICNS, 2005.

[10] A. Fox, S. D. Gribble, Y. Chawathe, and E. A. Brewer. Adapting to network and client variation using infrastructural process proxies: lessons and perspectives. Personal Communications, IEEE. 1998.

[11] Y. Chen. Detecting web page structure for adaptive viewing on small form factor devices. WWW, 2003.

[12] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. C. Rinard. Dynamic knobs for responsive power-aware computing. ASPLOS, 2011.

[13] W. Baek and T. M. Chilimbi. Green: A framework for supporting energy-conscious programming using controlled approximation. PLDI, 2010.

[14] T. F. Abdelzaher and N. Bhatti. Web content adaptation to improve server overload behavior. WWW, 1999.

[15] J. Yao, X. Liu, M. Yuan, and Z. Gu. Online adaptive utilization control for real-time embedded multiprocessor systems. CODES+ISSS, 2008.

[16] R. Reitano. Introduction to Quantitative Finance: A Math Tool Kit. 2010.

[17] I. Adan and J. Resing. Queueing Theory. 2001.

[18] R. Gullapalli, C. Muthusamy, and V. Babu. Control systems application in java based enterprise and cloud environments – a survey. Journal of ACSA, 2011.

[19] C. A. Yfoulis, and A. Gounaris. Honoring SLAs on Cloud Computing Services: A Control Perspective. ECC, 2009.

[20] X. Liu, J. Heo, L. Sha, and X. Zhu. Queueing-model-based adaptive control of multi-tiered web applications. IEEE Trans. on Network and Service Management, 2008.

[21] J. Hamilton. Blog article at http://perspectives.mvdirona.com/2009/10/31/thecostoflatency.aspx, 2009.

[22] W. Szpankowski. Bounds for queue lengths in a contention packet broadcast system. IEEE Trans. on Comm., 1986.

[23] H. Chen and P. Mohapatra, Session-based overload control in QoS-aware web servers. INFOCOM, 2002.

[24] L Cherkasova and P. Phaal, Session-based admission control: a mechanism for peak load management of commercial web sites. IEEE Trans. Comput., 2002.

[25] R Wang, Dara M. Kusic, N. Kandasamy, A distributed control framework for performance management of virtualized computing environments. ICAC, 2010.

[26] J. Leite, D. Kusic, D. Mosse, Stochastic approximation control of power and tardiness in a three-tier web-hosting cluster. ICAC, 2010.

[27] H. C. Lim, S. Babu, and J. S. Chase. Automated Control for Elastic Storage. ICAC, 2010.