Zeta: Scheduling Interactive Services with Partial Execution

Yuxiong He, Sameh Elnikety, James Larus, Chenyu Yan
Microsoft Research
{yuxhe, samehe, larus, chyan}@microsoft.com

Abstract

This paper presents a scheduling model for a class of interactive services in which requests are time bounded and lower result quality can be traded for shorter execution time. These applications include web search engines, finance servers, and other interactive, on-line services. We develop an efficient scheduling algorithm, Zeta, that allocates processor time among service requests to maximize the quality and minimize the variance of the response.

Zeta exploits the concavity of the request quality profile to distribute processing time among outstanding requests. By executing some requests partially (and obtaining much or most benefit of a full execution), Zeta frees resources for other requests, which might have timed out and produced no results. Compared to scheduling algorithms that consider only deadline or quality profile information, Zeta improves overall response quality and reduces response quality variance, yielding significant improvement in the high-percentile response quality.

We implemented and deployed Zeta in the Microsoft Bing web search engine and evaluated its performance in a production environment with realistic workloads. Measurements show that at the same response quality and latency as the production system, Zeta increases system capacity by 29% by improving both average and high percentile response quality. We also implemented Zeta in a finance server that computes option prices. In this application, Zeta improves average response quality by 28% and the 99-percentile quality by 80%. Using a simulation, we also compared Zeta to the offline optimal schedule and other scheduling algorithms. Although Zeta is only close to optimal, it provides better performance than prior algorithms under a wide variety of operating conditions.

Categories and Subject Descriptors D.4.1 [*Process Management*]: Scheduling

General Terms Algorithms, Experimentation, Performance

Keywords Deadline, Interactive service, Partial execution, Quality, Quality profile, Response time, Scheduling, Web search

1. Introduction

This paper shows how to improve the quality of an interactive web service's response by more effectively allocating processing time among competing service requests. We propose a new scheduling model for these services and describe and evaluate an efficient scheduling algorithm based on this model that increases overall result quality and reduces result quality variance. We implement the algorithm in two systems—the Microsoft

SOCC'12, October 14-17, 2012, San Jose, CA USA
Copyright © 2012 ACM 978-1-4503-1761-0/12/10. . . $15.00

Bing web search engine and a finance server—to demonstrate its benefits. We also use simulation to evaluate the scheduling algorithm under a wide range of operating conditions.

Many online services repeatedly execute tasks with three properties: (1) A request can be partially executed and will produce *partial results*. Given additional resources, such as processing time, the results will improve in quality. (2) Response quality improves with increasing resources but with diminishing returns, resulting in a *concave quality profile*. (3) The processing for a request is limited by time or resource constraints, e.g., it has a *deadline*.

We introduce a new scheduling model — Zeta to exploit these properties, which characterize important applications such as: (1) Web search: A web search engine receives queries from clients and returns the matched documents within a short deadline. A search query has multiple acceptable answers. With more processing time, the search engine will match and rank more web pages, producing a progressively better response. (2) Finance servers: Traders interactively submit requests to obtain an estimate of the price of financial derivatives. A finance server executes a computation, such as a Monte Carlo pricing algorithm. Increased processing time reduces the expected error of the calculation.

For interactive services with these properties, the objective of our scheduler is to improve both the average response quality and the high-percentile (e.g., 95%) response quality. Response quality is an application-specific metric that compares the result of partially executing a request against the results from full execution. Improved average quality allows an operator to: (1) target a given level of quality with fewer resources or (2) obtain better quality with the same resources. The high-percentile quality guarantees help ensure consistently high-quality results, even at the extremes of a service distribution. Many commercial services specify their Service Level Agreement (SLA) using both the average and high-percentile response quality [13]. For example, a web search engine may target a quality of 0.99 for at least 95% requests, which we call 95-percentile quality.

The Zeta scheduling model takes into account partial execution, request quality profile, and deadlines. Previously published algorithms do not consider all three properties. For example, several resource management models [14, 20, 23–25] do not consider partial execution. They either service a request in full or reject a request completely. Among resource management models that allow partial execution, QoS-adaptive systems [2, 19, 28] do not consider deadlines and soft real-time schedulers [7, 9, 10, 22] do not exploit concave quality profiles. Our experimental results show the importance of considering both deadline and quality profiles for web search and other interactive services.

From the optimization model, we derive the optimal offline schedule and a practical online algorithm. The online algorithm combines two techniques: (1) Equi-partitioning to allocate resources more equally among requests while respecting their deadlines. Equi-partitioning prevents long requests from starving short ones, which enhances overall response quality and reduces response quality variance. (2) Reservations, which execute long tasks completely when they will not affect short requests. Zeta *combines* these two algorithms based on load to maximize the execution time of each request subject to the constraint that other waiting requests should not miss their deadline.

Zeta scheduling is practical—we implement and deploy it in the Microsoft Bing search engine and evaluate it under a realistic workload in a production environment. Our experimental results show a significant improvement over the prior scheduler in both average response and 95-percentile quality. In particular, to meet the desired response quality and latency SLA, Zeta increases system capacity by 29%, or potentially reduces the number of servers by 22%. We also apply Zeta scheduling in a finance server that uses Monte Carlo methods to evaluate option prices. Each request is a task that estimates the price of a financial option and response quality is measured by the estimation error. Zeta improves average response quality by 28% and improves the 99-percentile quality by 80%.

In addition, we use simulation to evaluate Zeta in scenarios that cannot be easily implemented. We assess the benefits of the Zeta online scheduling algorithm by comparing it against the offline optimal and two online algorithms. We also evaluate Zeta using different shaped quality profiles. Our results show that the Zeta online

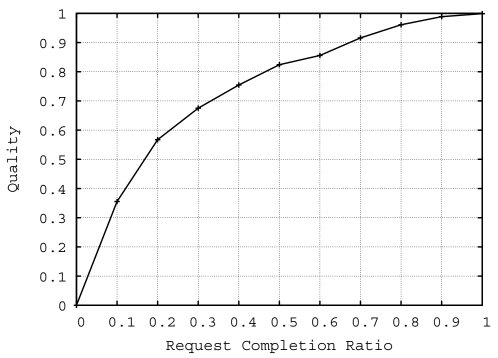


Figure 1. Measured quality profile in the search engine.

scheduler performs near optimal and produces better quality results than other algorithms under a variety of concave-like profiles.

The remainder of the paper is organized as follows. Section 2 and Section 3 describe Zeta scheduling model and algorithms. Section 4, Section 5 and Section 6 present Zeta implementation results in Bing, in a finance server, and our simulation results. Section 7 presents the related work and Section 8 concludes the paper.

2. Zeta Scheduling Model

This section defines and motivates the three characteristics of the Zeta scheduling model. It also presents a formal model of the scheduling problem.

2.1 Model Characteristics

(1) Partial execution. Partial execution exploits the possibility of trading fewer resources for degraded results. Without partial execution, a scheduler can either execute a request fully or reject it. By contrast, partial execution opens the possibility that some requests are processed partially, returning an approximate result, which is a favorable trade-off in many situations in which resources are in short supply. For many computations, better estimates are obtained with more processing, but inexact results are acceptable, even if slightly less desirable. For example, in web search, if a request does not hit in the query cache, the search engine scans its inverted index looking for web pages that match the requested keywords and ranks the matching webpages. The more time the server spends matching and ranking the web pages, the better the results. If the search engine does not finish processing all matched web pages, it can still return the best results found so far.

(2) Concave quality profile. With partial execution, the relationship between the quality of the result and the amount of computational resources has a concave shape. The concavity of the quality profile is the result of diminishing returns and the iterative nature of approximation algorithms.

We measured the response quality profile in Bing. Figure 1 shows a concave quality profile produced from 200K queries in a production trace. The response quality compares the set of documents returned in the test against a golden set of base results obtained with full execution (as explained in Section 4). The x-axis of Figure 1 is *request completion ratio*, which is calculated as the actual processing time divided by its full processing time. The y-axis is the average response quality. The figure demonstrates that the quality profile is monotonically increasing and concave. The concavity comes from the design of the server, in which the inverted index lists important (e.g., popular) webpages first; therefore webpages matched earlier are more likely to rank higher and contribute more to response quality.

Diminishing returns are common in many domains. Another example is finance computations, where increasing the number of samples in Monte-Carlo computations reduces the result uncertainty. However, the incremental gain of an additional sample becomes smaller when the number of samples becomes larger.

Algorithm 1 - OPTIMAL

Inputs:
$J = \{J_i | i = 1, \ldots, n\}$: set of jobs
$S = \{S_k | k = 1, \ldots, 2n-1\}$: set of segments
w_i, f_i: service demand and quality function of job J_i

Variables:
p_{ik}: processing time of job J_i at segment S_k.
p_i: total processing time of J_i.

Optimization model:

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

subject to:
Resource availability: $\sum_{i \in J} p_{ik} \leq |S_k|$ for all $S_k \in S$
Total processing time: $\sum_{k \in S} p_{ik} = p_i$ for all $J_i \in J$
Service demand: $p_i \leq w_i$ for all $J_i \in J$
Validity: $p_{ik} = 0$ if J_i is not valid at S_k

(3) Deadline. Users expect timely responses from most interactive services. Long response times are unacceptable because they cause user dissatisfaction and revenue loss [16]. Timing constraints are typically expressed as deadlines.

2.2 Problem Formulation

We define the scheduling problem as follows. There is a set $J = \{J_1, J_2, \ldots, J_n\}$ of n jobs. Each job $J_i \in J$ is characterized by an *arrival time* r_i, *deadline* d_i, and *service demand* (aka *total work*) w_i. The *deadline interval* is the difference between the deadline and arrival time ($d_i - r_i$). Let p_i denote the *processing time* job J_i receives during $[r_i, d_i]$. It may not be processed to its full service demand, that is, partial execution is allowed with $p_i \leq w_i$. A job can be preempted; it can be suspended and resumed later.

A *quality function* $f_i : R \rightarrow R$ maps the job completion ratio (processing time / service demand) to a quality value gained by executing the job. The objective of the scheduler is to maximize the total quality gained by executing all jobs in the job set, i.e., $\sum_{i=1}^{n} f_i(p_i/w_i)$.

3. Zeta Scheduling Algorithms

This section presents an optimal offline model and an online algorithm for Zeta scheduling.

3.1 Optimal Offline Model

We derive an offline optimization model to maximize request quality, exploiting complete request information including future arrivals. Although some of this information is unknown in practice, the offline optimal often inspires the design of online algorithms and serves as a performance reference.

The main challenge in producing an optimization model for this scheduling problem is that a job can be preempted and resumed an unbounded number of times. Thus, the job needs an unbounded number of variables to represent the processed intervals.

We solve the problem by limiting the number of preemptions each job may incur without affecting the optimal quality. Define an interval in which there is no job arrival or expiration as a *segment*. In any segment, a job only needs to run at most once. If a job runs multiple times in a schedule F, we can produce an equivalent schedule F' that combines the job processing time within the segment in schedule F into one continuous processing period in schedule F'. This will not violate the deadline of any job because there is no job arrival or expiration within the segment. With this observation, given n jobs, their arrival time and deadline divides the available time interval into $2n-1$ consecutive segments denoted as $\{S_k|k=1...2n-1\}$. Each segment S_k is a time interval $S_k=[a_k,b_k]$, where the starting time a_k and ending time b_k of S_k correspond to arrival time or deadline of jobs. There is no job arrival or job expiration within a segment and each job is processed at most once within a segment. This leads to the following optimization model.

OPTIMAL (Algorithm 1) presents an offline optimal solution for the problem. There are n jobs and $2n-1$ segments. For each segment S_k, a job J_i is defined as *valid* iff $S_k=[a_k,b_k]\subseteq[r_i,d_i]$, i.e., $r_i\leq a_k$ and $b_k\leq d_i$; otherwise, the job is invalid. Within each segment S_k, the status of any job does not change: a job is either valid at any time within the segment or not valid through the entire segment. The optimization model captures four sets of constraints:

- **Resource availability:** Total processing time of jobs within a segment is bounded by the segment length, i.e., $|S_k|=b_k-a_k$ for $S_k=[a_k,b_k]$.
- **Total processing time:** Total processing time of a job is the sum of its processing time over different segments.
- **Service demand:** Total processing time of each job is no more than its service demand.
- **Validity:** A job can only be processed in its valid segments.

In this optimization model, all constraints are linear, and the objective is to maximize the summation of the quality functions. If the request quality functions are arbitrary nonlinear functions, the model becomes a general optimization problem with no efficient optimization method. However, if all quality functions f_i are concave, this is a convex optimization problem, which has nice properties. An optimal solution exists and can be obtained by standard convex optimization packages. As we discuss in Section 2, in many practical problems the quality profile is concave.

3.2 Online Algorithm

The offline optimal produces schedules with the highest possible quality, however, it is not usable online because it requires complete information of requests including future arrivals. This section presents an efficient online algorithm for servers. The server environment is an instance of the general Zeta scheduling model with the following requirements and assumptions: (1) The exact service demand and quality profile of requests are unknown. (2) Preemption of a request incurs a relatively large context switch overhead as request service demand is often only tens of milliseconds. (3) Requests are not differentiated, thus they all have the same deadline. (4) The online scheduler should be computationally efficient with a small overhead.

We develop an efficient online algorithm Zeta-Online in four steps. First, we introduce the challenges that the online algorithm must address. Second, we present two techniques to improve response quality and reduce response quality variance. Third, we show how to combine these two techniques and derive the Zeta-Online algorithm. Finally, we present a brief performance comparison of Zeta-Online with alternative algorithms.

3.2.1 Challenges

To provide an efficient online scheduler in interactive server systems, we address the following challenges:

Challenge 1: Future service demand and the quality profile for each request are typically unavailable online. *Solution:* Zeta-Online uses the expected (or average) quality profile and service demand. For example, Figure 1 shows the average quality profile for a web search request in Bing.

Challenge 2: Interactive requests must be answered in bounded time. *Solution:* To incorporate a deadline constraint, Zeta-Online processes requests in an earliest-deadline-first order. When requests are not differentiated and have the same deadline interval, then Earliest Deadline First (EDF) is equivalent to serving requests in FIFO order.

Challenge 3: Context switches can be expensive for interactive requests. In practice when a job's working set is large, a context switch is costly because of cache warm-up. It may take a few hundred microseconds to more than a millisecond to bring data into the cache[21]. Since the service demand may only be a few tens of milliseconds, frequent context switching is expensive. *Solution:* To reduce context switch overhead, Zeta-Online processes each request once.

In our server environment in which all requests have the same deadline, as we show in Lemma 1, any feasible schedule that can arbitrarily preempt and resume a job's execution can be transformed into an equivalent FIFO schedule that runs each job once without context switching while achieving the same total quality. This is an important result: Serving requests in FIFO order reduces context switching overhead without compromising response quality.

LEMMA 1. *Suppose that requests have the same deadline interval. For any feasible schedule with arbitrary preemption and resume of a job, there exists a feasible schedule that executes all jobs in the FIFO order with the same overall quality.*

Proof. Given any feasible preemptive schedule, consider any two jobs J_i and J_j that do not follow the FIFO order. Without loss of generality, we assume that J_i arrives first, and thus has the earlier deadline. Without affecting the execution of other jobs, we can execute J_i before J_j, both for the same amount of processing time as the original schedule, hence the overall quality gained is the same. The new schedule is also feasible since the deadlines of both jobs are not violated. Applying the same argument to all pair of jobs in the original schedule gives us a FIFO schedule that processes each job once only and gives the same response quality. □

3.2.2 Two Techniques to Improve Response Quality and Reduce Variance

In additional to these challenges, the online algorithm should increase total response quality and to reduce response quality variance. Both improvements are needed to increase the high-percentile quality, which is part of the SLA of many commercial services. To this end, we introduce two techniques: EQ and RESV.

(1) Equi-Partitioning (EQ) applies the following intuition: When requests are competing for resources, it is desirable to run the portion of each request that produces a large gain. Under an expected concave quality profile, this means giving each request equal processing time. EQ partitions the available resources among requests by assigning the same processing time to each request so long as they meet their deadlines. This strategy also prevents long requests from starving short requests. Moreover, since EQ tries to give all requests the same amount of processing time, it reduces the processing time variance among different requests, and therefore reduces variance in response quality.

(2) Reservation (RESV) reserves the expected processing time for waiting requests in the queue and assigns the remaining processing time to execute the current request. RESV prevents a long request from starving subsequent short ones, while allowing the current long request to obtain more resources, up to the point at which it affects subsequent requests.

The following example shows that, in online environment with unknown service demand and without preemption, both techniques are required to achieve high response quality.

Example. Assume that two jobs arrive at the same time, with 120ms deadline, and we process each job only once. After we run the first job for 60ms, should we continue the first job or move to the second? EQ assigns each request 60ms, and therefore stops the first job to process the second. It is possible, however, that the second job only needs 30ms and we will waste 30ms that could have been used for the first job. A better decision is to look at the expected service demand of requests. If the demand is large (above 60ms), we should perform EQ. Otherwise, we use RESV to reserve the expected processing time for the second job. For example, if expected service demand is 30, we let the first job run till completion or $120 - 30 = 90$ms, whichever is earlier, and reserve 30ms for the second. This example motivates why both EQ and RESV are necessary.

3.2.3 Zeta-Online Algorithm

This section describes how to combine EQ and RESV in the Zeta-Online algorithm.

Intuitively, the selection of EQ and RESV should depend on the system load. Under heavy load, EQ performs well because each job should get an equal amount of execution time to improve quality. On the other hand under

Algorithm 2 - Zeta-Online (Jobs[] $queue$, Job $active$, double $curtime$, double $meanDemand$)

Inputs:

$active$: current running job;
$queue$: list of ready jobs (including active job at $queue[0]$ if $active$ is not null);
$curtime$: current time stamp;
$meanDemand$: mean service demand of jobs;

Pseudo code:

```
1: $lastJob = queue[queue.size() - 1]$
2: $queueLen = queue.size()$
3: $if(active \neq null)$
4:   $then\, timeBase = active.startProcessingTime$
5:   $else\, timeBase = curtime$
6: $timeAvail = lastJob.deadline - timeBase$
   //  Equi-partitioning
7: $EQ = timeAvail/queueLen$
   //  Reservation
8: $RESV = timeAvail - (queueLen - 1) \times meanDemand$
9: $queue[0].processingTime = max(EQ, RESV)$
```

light load, RESV performs well because it gives long requests the chance to finish when they do not affect short ones. One approach is to select an arrival rate threshold to switch between EQ or RESV. This approach is undesirable because it cannot capture the transient underload and overload situation, which occurs often due to stochastic nature of the request inter-arrival and service processes. The instantaneous load oscillates even if the average arrival rate and service demand do not change, and as queuing theory shows, the length of the request waiting queue changes due to the transient load changes. Coarse-grain switching between EQ and RESV cannot capture those changes.

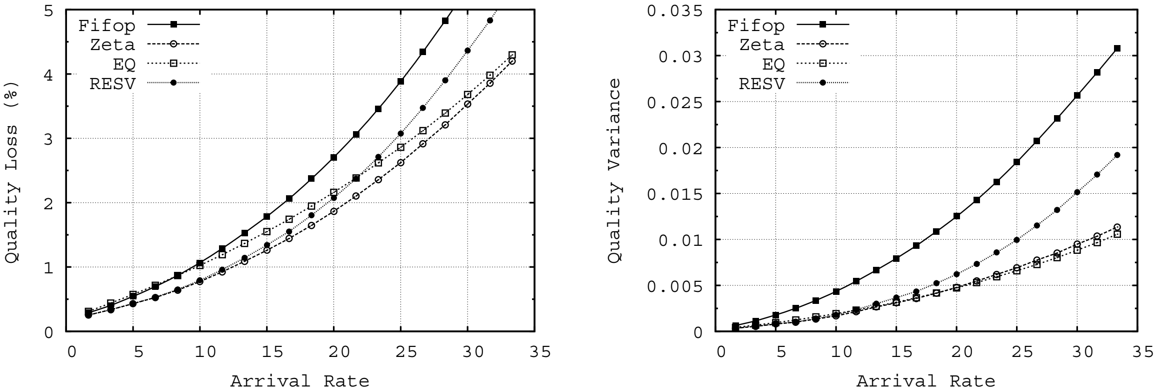
Zeta-Online does not use a threshold to select between EQ and RESV. When system is lightly loaded, we want to estimate processing time using RESV; at this point, its processing time produced is larger than the one produced by EQ. When system is heavily loaded, we want to use EQ; at this point, its processing time value is larger than the one produced by RESV. Therefore, Zeta-Online selects the larger value produced by the two policies. Zeta-Online makes a fine-grain selection between the two techniques, which incoporate the queue length and average service demand, good indicators of instantaneous system load. We show in Section 3.2.4 that Zeta-Online achieves higher quality than EQ alone or RESV alone across all loads, as it responds to the instantaneous load.

Zeta-Online is presented in Algorithm 2. Zeta-Online assigns processing time to a request based on the number of ready requests in the queue ($queueLen$) and the available processing time of all requests in the queue ($timeAvail$). It computes the assigned processing time of the current request based on EQ (in Line 7) and RESV (in Line 8). Rather than using a load threshold to choose the result from EQ or from RESV, Zeta-Online combines the two techniques seamlessly by selecting the larger processing time they calculate (in Line 9).

3.2.4 Overview of Algorithm Performance

This section presents simulations that compare the performance of Zeta-Online against its constituent algorithms EQ and RESV and against a classic server scheduling algorithm.

Figure 2 compares the response quality and variance of Zeta-Online against other three algorithms: (1) FifoP (First-In-First-Out with Partial results) represents a classic server scheduler that processes requests in first-in-first-out order. A request runs until its completion or its deadline. At the deadline, processing is terminated,



(a)

(b)

Figure 2. Overview of algorithm performance using simulation results.

returning a partial result. It is equivalent to Earliest-Deadline-First (EDF) in our environment. FifoP is our baseline algorithm. (2) EQ (Equi-Partitioning) assigns available processing time equally among ready requests in the queue. It is equivalent to Zeta-Online without the RESV technique. (3) RESV (Reservation) reserves the expected processing time for waiting requests in the queue, granting current request the remaining time. It is equivalent to Zeta-Online without the EQ technique. The algorithms are compared by simulation, using requests with the Bing quality profile (Figure 1). We defer details of the simulation setup to Section 6.2.

Figure 2(a) and Figure 2(b) compare the quality loss and quality variance of the four algorithms at different arrival rates. There are three observations from the results:

(1) Zeta-Online incurs lower quality loss (achieves higher quality) than FifoP. The reason is that FifoP is only deadline aware, but it does not exploit the concavity of the quality profile. Consider an example: two requests with 100ms service demand, 120ms deadline, and the quality profile in Figure 1. FifoP gets total quality of 1.56 by executing the first job for 100ms and the second job for 20ms. Zeta-Online can do better by assigning the first job 60ms, and allocating the remaining time to the second one, resulting in total quality of $0.88 + 0.88 = 1.76$. This improvement comes from using the concavity of the quality profile.

(2) Figure 2(b) shows that Zeta-Online not only improves response quality but also reduces the quality variance compared to FifoP. The low variance of Zeta-Online comes from EQ, which assigns the same processing time to requests, reducing the processing time variance among requests. The variance of FifoP is significantly larger as a long request may starve the execution of later requests, producing a larger variance in response quality.

(3) Using EQ or RESV alone is not enough. As illustrated at Section 3.2.2, EQ prevents long requests from starving short ones by giving all requests the same amount of processing time. This works well during heavy loads, but under light load, it may truncate long requests unnecessarily even when the long requests have a chance to finish. Therefore, EQ does not perform well under light and moderate load as shown in Figure 2(a). By contrast, RESV gives long requests a chance to finish under light load by reserving their expected service demand for later requests. When load is heavy, RESV over-reserves for these requests since they should receive less than the expected service demand. The figure shows that RESV does not perform well under heavy load. Comparing Zeta-Online to EQ and RESV, Zeta-Online has lower average quality loss than any of them for all loads.

Moreover, as discussed in 3.2.3, if we apply a coarse-grain switch that selects EQ or RESV based on the arrival rate, the quality loss curve would be the minimum of EQ and RESV. This coarse-grain switch underperforms the fine-grained selection used in Zeta-Online, especially in the moderate-high load from 20 to 30 QPS. The improvement of Zeta-Online comes from appropriate selection of techniques at transient load changes.

In summary, Zeta-Online considers partial execution, concave quality profile, and deadlines by seamlessly combining two techniques EQ and RESV to improve response quality and reduce response variance. Moreover, it possesses three desirable properties. It does not require precise information on the service demand and quality profile of requests. It does not incur preemption overhead. And, it is computationally efficient with the computational complexity $O(1)$, independent of the number of ready jobs. We further evaluate Zeta's performance in two actual systems in Sections 4 and 5. To simplify the presentation, we refer Zeta-Online as Zeta for the remainder of the paper.

4. Web Search Engine

In this section we implement and evaluate Zeta in Microsoft Bing, which is the second largest commercial web search engine, to show the benefits of Zeta in a production environment.

4.1 Bing Search Engine

We focus on the index serving part that processes user web search queries (interactive processing), rather than on the web crawler and indexer (batch processing). When Bing receives a request, if the request hits in the cache, the results are returned, otherwise, the request is forwarded to the index serving system. The index serving system accepts user queries, searches its index for web pages that match the query, and ranks matched documents to return the top N matches. Web search is a best-effort application where the result quality improves with the increased number of documents retrieved and ranked. It is also interactive because the results need to be returned within a given time budget; otherwise, it significantly affects user experiences. These two properties of web search make it a suitable candidate for Zeta.

Architecture of the Index Serving System. The index serving system is a distributed service consisting of aggregators and index servers. When a request arrives and its results do not exist in cache, it is assigned to an aggregator. The aggregator sends the request to the index servers. Because the web index contains billions of documents, the index is partitioned and processed by hundreds of index servers. Each index server returns its matched results to the aggregator within 150ms; the aggregator drops late responses. The aggregator receives the results from each index server and returns the top N documents. Due to the large number of index servers, several levels of aggregators are used. The dominant component is the index server and it consumes the majority (over 90%) of hardware resources. Bing search infrastructure uses acceleration techniques, including caching. However, such techniques are orthogonal to our work.

4.2 Implementation in Index Servers

When an index server receives a search query, it searches its index to produce a list of documents matching the keywords in the query. It then ranks the matching documents one by one in a loop, which we call ranking loop. It is the most time consuming part of the index search because the ranking function extracts and computes many features of the documents. Without Zeta, the index server uses a FifoP scheduler: requests are processed in first-in-first-out order; a request is processed until its completes or the deadline expires. In the latter case, it returns a partial result.

We implemented Zeta-Online and use it to assign processing time for each request. To do so, we add a new termination condition to the index server. This termination condition compares, at the start of each iteration of the ranking loop, the elapsed processing time of a request with its assigned processing time. If the elapsed processing time is greater than or equal to the assigned processing time, the Zeta-enabled index server terminates the ranking process and returns the top N ranked results.

Zeta-Online can be extended to multicore servers by changing the queue length value ($queueLen$) to reflect the expected queue length for each core. The mean service demand $meanDemand$ is computed online in a recursive manner as a moving average: $meanDemand = (1 - \alpha) \times meanDemand + \alpha \times p_i$, where p_i is the processing time of the last processed request i and α is a constant multiplier. We use $\alpha = 0.05$ in our implementation. Zeta only required a small change to the existing server, suggesting that Zeta is practical and can be applied in large-scale interactive services.

4.3 Experimental Evaluation

We evaluate the performance of Zeta in Bing by comparing it to Traditional (the system before Zeta) using production workloads. Zeta and Traditional are identical software and hardware, except for the request scheduling. Traditional uses FifoP scheduling as in the production servers and Zeta uses Zeta-Online. We perform both standalone (single machine) and cluster experiments but report only cluster results here.

The cluster experiments are conducted on 800 index servers, each hosting a portion of the web index. In the cluster experiments, each index server returns its top 5 results, and top-level aggregator sorts and returns the top 10 results aggregated from all index servers. The deadline of the requests is 150ms; a late response from an index server is dropped by the aggregators. We play requests from a production trace of user queries of Bing. Each data point (per QPS) in the experiment runs for 30 minutes, and the number of requests depends on the query rate (QPS). The requests follow a Poisson arrival and their average service demand is 10.5ms. To protect commercial business interests, we do not disclose the hardware configuration of the index servers.

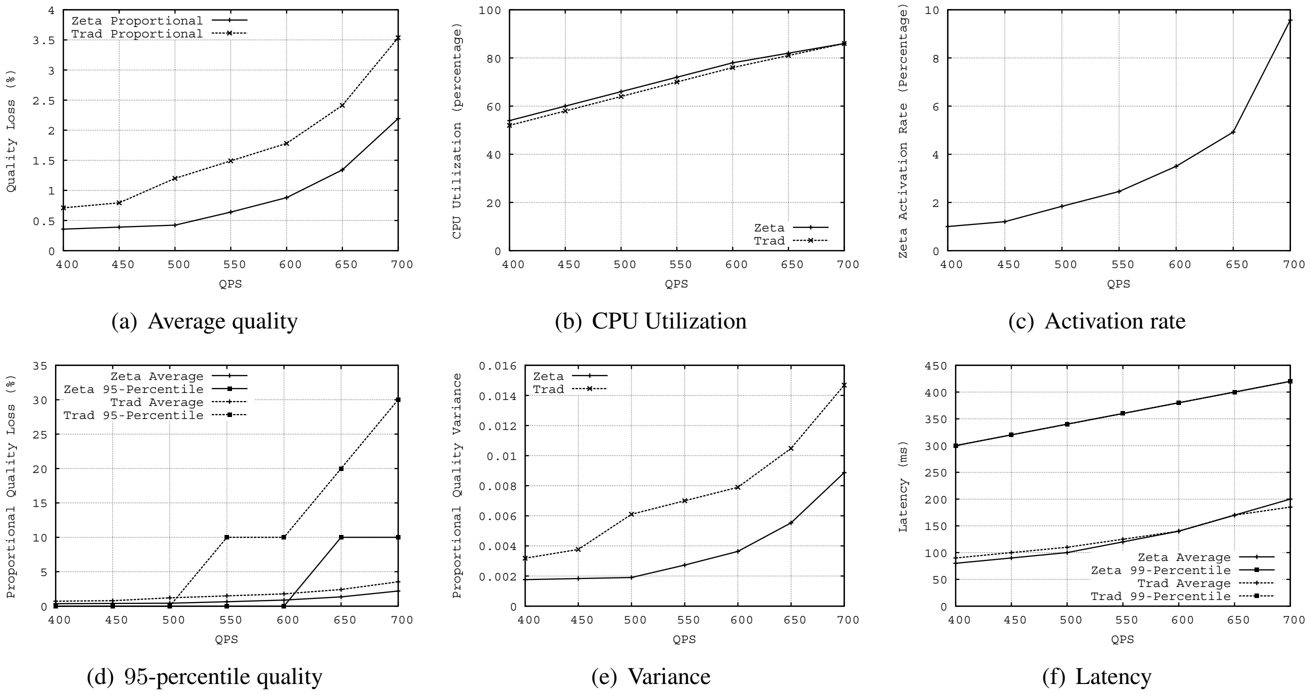
Performance Metrics

Response Quality. To score the quality of a response to a search request, we compare returned documents in the response to the documents in the *base* results of the query when it is processed completely. We use *proportional quality*, which gives each of the top $N = 10$ documents the same weight. For example, when there are 8 matches with the base, the quality is $8/10$. Proportional quality is one way to measure response quality, and we also discuss other quality metrics in Section 4.4.4. We present quality loss values in the figures. Given average quality of requests ρ, the corresponding quality loss value is $1 - \rho$.

Response Latency. We measure query latency from the time that the top-level aggregator receives the query to the time the aggregator responds to the client.

CPU Utilization. We report average CPU utilization of all the index servers in the cluster.

4.4 Cluster Experiments



(a) Average quality

(b) CPU Utilization

(c) Activation rate

(d) 95-percentile quality

(e) Variance

(f) Latency

Figure 3. Web search evaluation results.

This section presents four sets of experiments with their results highlighted as follows:

- **Average response quality.** Zeta improves average response quality over Traditional and effectively increases the system capacity. Under SLA quality target of 99%, Zeta achieves a 29% capacity improvement, a potential savings of 22% of servers.
- **High-percentile response time.** Zeta achieves much higher 95-percentile quality as a result of improved average quality and reduced quality variance.
- **Response time.** The response time of Zeta is similar to Traditional at any given load. Zeta does not trade response time for quality.
- **Sensitivity study on quality metrics.** Zeta consistently outperforms Traditional regardless of the quality metric.

4.4.1 Average Quality Results

Figure 3(a) shows that Zeta offers better quality than Traditional. The x-axis is the load from 400 and 700 QPS, and the y-axis is the quality loss. At 600 QPS, Traditional has quality loss of 1.8% and Zeta reduces it to 0.9%. By cutting quality loss in half, Zeta reduces the number of unsatisfied requests by half as well, improving customer satisfaction. Without Zeta, to achieve the same quality loss at 0.9%, Traditional can only sustain 450 QPS so it requires $(1/450 - 1/600)/(1/600) = 33\%$ more servers, a large expense at this scale.

One may wonder why it is so costly to improve quality by only 0.9% (or equivalently reduce quality loss by 0.9%) when the existing quality is already high. Since both request arrival and service demand are stochastically distributed, transient overload periods exist although with small probability. In order to achieve very high quality, the system has to be provisioned to handle the transient overload instead of the average load, which requires many more additional servers. Therefore, at a high quality range, the number of servers needed to improve quality increases much faster than the improvement in quality: a small improvement in quality requires a large number of servers.

To explain the reasons for quality improvement, we start from the quality profile. Figure 1 shows that the measured quality profile is concave. Traditional uses FifoP to serve requests one by one. Here, long requests may starve short requests even though the quality gain during the late stages of long requests is small. Zeta prevents these long requests from starving short ones by terminating them earlier. Figure 3(c) shows Zeta activation rate, which indicates the frequency of requests terminated early due to expiration of the processing time assigned by Zeta. At 600 QPS, 3.5% requests terminated early. Terminating long requests early, at a small loss in quality, frees processing time for other requests with potentially higher quality gain. This is the principal contribution of Zeta.

At the same quality, Zeta produces higher throughput and higher CPU utilization (CPU utilization is shown in Figure 3(b)). For example, at a 1% quality loss Traditional supports 475 QPS with 64% CPU utilization, and Zeta sustains 613 QPS with 80% utilization, showing a $(613 - 475)/475 = 29\%$ capacity improvement.

4.4.2 High-Percentile Quality Results

A service attempts to produce high-quality responses to every query. We continuously monitor the performance of our search service using both the average and 95% percentile of response quality.

Figure 3(d) shows the average and 95-percentile quality loss[1]. At load QPS=600, Zeta reduces average quality loss from 1.8% to 0.9%, and reduces the 95-percentile quality loss from 10% to 0%. Zeta exhibits significant improvement on the 95-percentile quality because it reduces quality variance.

Even when Zeta and Traditional produce similar average quality, Zeta has better 95%-percentile quality because of smaller variance. Figure 3(e) shows that Zeta has smaller variance than Traditional. Smaller variance improves the high-percentile quality because, intuitively, if a set of values has a smaller variance, then the data are more concentrated near the mean, reducing the frequency of outliers. Zeta prevents long requests from

[1] The 95-percentile response quality is discrete because the proportional quality measurement of the top 10 results can only be multiples of $1/10$ from 0 to 1.

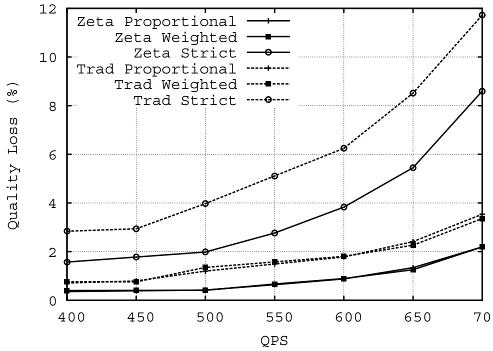


Figure 4. Cluster test quality comparison with different quality metrics.

starving short ones, which tends to equalize the assigned processing time of requests and to reduce response quality variance on each server.

The aggregated response from all index servers has lower variance as well. Assume each request is processed on a set G of index servers, where each server hosts a partition of web index. For simplicity, assume that the top N results come from different servers: this is likely true since it takes thousands of servers to host the entire web index. Let S represent the set of these servers with the top N results, and $|S| = N$. The quality of the aggregated result depends on the quality of these N servers. Using proportional quality, each server contributes to $1/N$ of the total quality. Denoting X_i as the quality of a single server, the aggregated quality $Y = \sum_{i \in S} X_i/N$. Assuming index servers are identical and independent, $var(Y) = \sum_{i \in S} var(X_i)/N^2 = var(X_i)/N$. The variance of the aggregated result is a function of the variance of single servers.

4.4.3 Response Time

As shown in Figure 3(f), average and 99-percentile response time of Zeta and Traditional are similar. Zeta does not trade worse response time to improve response quality, instead it allocates computational resources more efficiently among the requests.

4.4.4 Sensitivity to Quality Metrics

Measuring the quality of a web query response is an active area of research with various performance and relevance metrics [12]. Besides proportional quality, we consider two alternatives, strict and weighted quality.

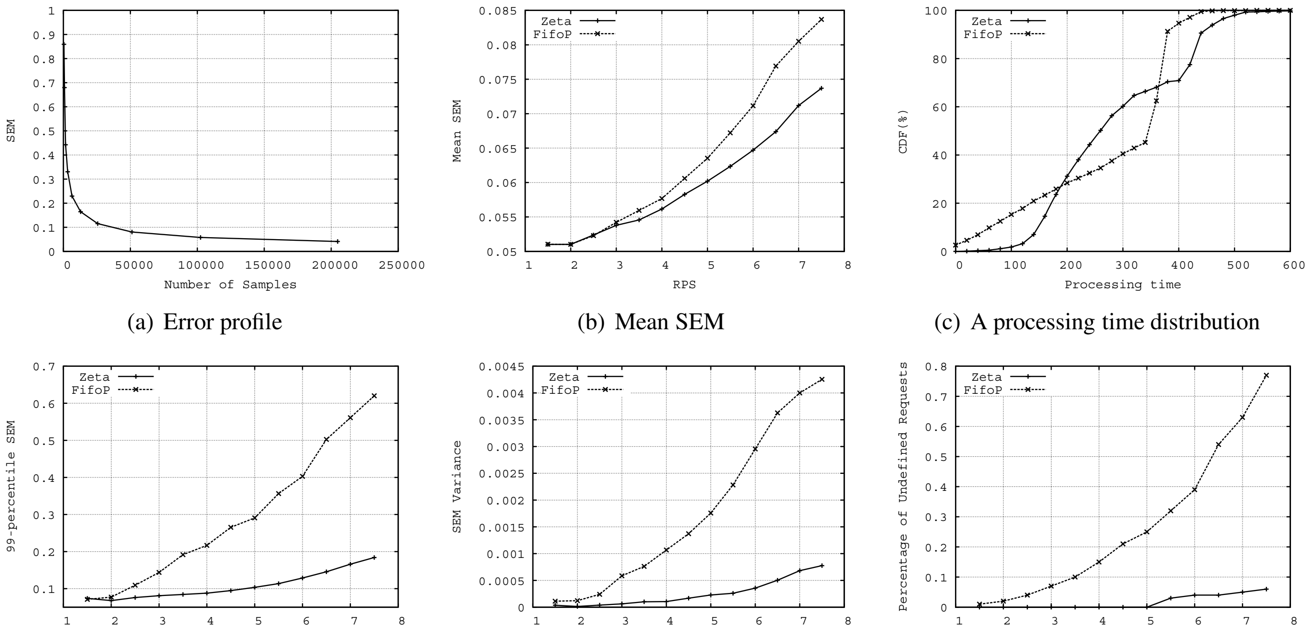
- Strict quality: if the query test results are an *exact match* of its base results, the strict quality is equal to 1; otherwise it is 0. In our experiments, the query response includes its top 10 results, therefore, the response must match the exact 10 results and in the original order.
- Weighted quality: higher ranking results carry more weight than lower ranking results and therefore order of results is important. We assign weights using an exponential scale, giving the top ranking result $1/2$ of total weight, the second result $1/4$ of the weight and so on.

Strict quality demands exact match and no partial response is accepted. Proportional quality and weighted quality accepts partial results. Since for web search, every matching document carries certain value to the users, weighted and proportional quality are more appropriate metrics. For completeness, however, we present the results for all these metrics.

Figure 4 shows that Zeta produces better quality than Traditional for all three quality metrics. Using weighted or proportional quality, Zeta supports a 29% capacity improvement over Traditional.

For the strict quality, Traditional supports 475 QPS with quality loss 3.5%. Zeta offers this quality at 570 QPS, which indicates a 20% capacity improvement. Zeta also reduces quality loss to 1.8% at the same load, a 48% reduction in quality loss. Zeta's improvement in strict quality is slightly less than weighted and proportional quality because Zeta embraces adaptive execution. Zeta still outperforms Traditional because Zeta cuts long requests so more short requests get a chance to run, which improves the total quality. The results show that Zeta's quality improvement is consistent, not depending on specific ways of measuring response quality.

5. Finance Server



(a) Error profile

(b) Mean SEM

(c) A processing time distribution

(d) 99-percentile SEM

(e) Variance

(f) Number of undefined requests

Figure 5. Finance server evaluation results.

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. These computations involve three main techniques: Monte Carlo methods, Black-Scholes, lattice-based computation. Among them, Monte Carlo methods are widely used for analyzing complex derivatives that are difficult to value using other techniques. The calculations are computationally intensive and rely on repeated random sampling to compute results.

This section applies Zeta to improve the performance of an option pricing server that uses Monte Carlo methods to price complex path-dependent options. The Monte Carlo tasks are time-bounded. Traders often wait for no more than a few seconds to get the results and perform online trading. Moreover, they support partial results: with more samples, the processing time is longer and the estimated price gets closer to the real price. These characteristics make it a suitable candidate for Zeta. Our experimental results show that Zeta improves average response quality by 28% and improves the 99-percentile quality by 80%.

5.1 Quality Metric

The result quality is often measured by a statistical metric called standard error of mean (SEM). We use a standard way to compute it which we explain here in three steps: (1) define SEM, (2) show its value (which we do not know), and (3) describe how to estimate it. (1) SEM by definition is the standard deviation of the sample mean to the population mean. It indicates how well the sample mean estimates the population mean. (2) The real SEM value is equal to the population standard deviation (denoted as σ) divided by the square root of the sample size (denote sample size as n), i.e., real SEM $= \sigma/\sqrt{n}$, however, the population standard deviation is often unknown in practice. (3) To estimate SEM we estimate σ using the sample standard deviation (denoted as s). Then we have estimated SEM using sample standard deviation divided by the square root of the sample size, i.e., SEM $= s/\sqrt{n}$, which we use to compute SEM in our experiments. The smaller SEM is, the closer the estimated price to with the real price. Although the real price is usually unknown in practice, SEM can be used to calculate a good approximation to a confidence interval. The goal of the server is to minimize the average and high-percentile SEM value for all requests so the estimated prices are closer to the real prices.

Notice that when the sample size is very small, the SEM estimator is statistically unreliable because $s/\sqrt{n}$ is subject to estimation error and can be far from the real SEM $\sigma/\sqrt{n}$. Therefore, we consider a request to be unprocessed if it does not get any processing time or if the processing time is too small to produce any statistically meaningful solution (we use 200 samples in the experiments). We report two numbers, the mean SEM for completed requests and the number of unprocessed requests.

5.2 Experimental setup

Our pricing server in experiments uses Monte Carlo methods to price Asian options [6, 11]. For Asian options the payoff is determined by the average underlying price over some pre-set period of time, therefore, the price is path-dependent. Closed form expressions for the option price are not available for this model. Although our experiments price Asian options, Monte Carlo methods to evaluate other financial derivatives are similar.

We implement and compare two schedulers FifoP and Zeta-Online; all other software and hardware are same. The hardware is a dual-core Intel machine with CPU speed 3.17 GHz and memory 4 GB. Each request executes a classic Monte Carlo Asian option pricing algorithm [6] to estimate the option price. The input parameters of a request include interest rate r, strike price K, dividend yield δ, volatility σ_y, total duration T and period m. In our test, all requests share the same total duration and period value; we use some typical Asian option value $T = 1$ representing 1 year as the duration and $m = 1/12$ representing 1-month period. Requests are evaluated under various economic scenarios so they have different interest rate, strike price, dividend yield and volatility; an example of these inputs is as shown in [6]: $r = 0.1$, $K = 100$, $\delta = 0.03$, $\sigma_y = 0.25$.

The requests follow Poisson arrival and the deadline is 1 second. We set SEM target 0.05: when a request's SEM reaches 0.05 or lower, we consider it fully evaluated and terminate the request. Otherwise, the request is partially evaluated. Request service demand is decided by the number of samples that is closely related to SEM target, and path length of each sample that is decided by T/m. The requests in our test have the same SEM target and $T/m = 12$, so their service demands are the same about 320ms.

5.3 Request Error Profile

Figure 5(a) shows the request error profile with increasing sample size. 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. The intuition behind the convex error profile is that SEM is measured as sample standard deviation divided by the square root of the sample size, i.e., SEM = $s/\sqrt{n}$. When the sample size is fairly large, the change of sample standard deviation is small and the square root of the sample size is the dominating term. The convexity of $1/\sqrt{n}$ leads to the convexity of SEM. Moreover, minimizing SEM with a convex error profile is equivalent to maximize quality with a concave quality profile, so Zeta is still applicable.

5.4 Performance evaluation

We compare Zeta with FifoP with respect to mean, variance, and high percentile SEM values of all requests. Our experimental results suggest that Zeta outperforms FifoP over all of these metrics. In particular, Zeta significantly reduces the variance (by over 5 times at high load), which leads to a big reduction at the high-percentile error.

5.4.1 Mean SEM

Figure 5(b) compares Zeta with FifoP with respect to mean SEM values. The results show that Zeta reduces mean SEM under moderate and heavy load, and with the increase of load, the gap between Zeta and FifoP becomes larger. This is consistent with what we observed at Search server: Zeta takes advantage of the concave quality profile (convex error profile) to improves quality (reduce error) through better scheduling. Since the error profile is convex, Zeta assigning similar processing to requests through equal-partitioning and reservation helps to minimize the mean SEM. It is demonstrated at Figure 5(c), which presents the processing time distribution of Zeta and FifoP at arrival rate 6 RPS. We can see that there are more requests with very small processing time (< 100ms) at FifoP than Zeta, which contributes to larger mean SEM.

5.4.2 High-Percentile SEM

Zeta significantly reduces the SEM variance: the reduction of variance is 5 times or more for most of the load (as shown in Figure 5(e)). The reason behind reduction of variance is that when requests are competing for resources Zeta tries to assign each request an equal amount of processing time while FifoP may complete some requests but leave other requests unprocessed or processed with little time. Therefore, requests scheduled by Zeta tend to have similar quality values and smaller variance than FifoP.

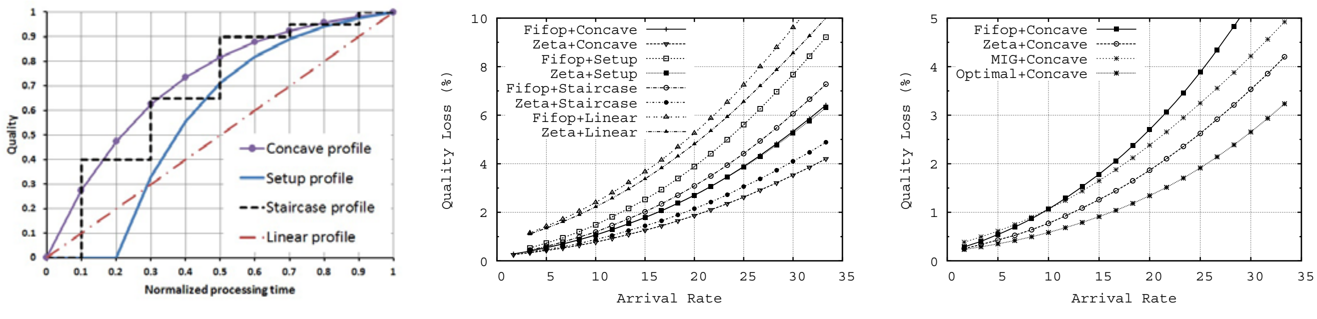
Smaller variance has important implications on reducing high-percentile error (or improving high-percentile quality), which are crucial for many commercial applications. To look at the impact of variance on high-percentile SEM, we can compare the error reduction of Zeta at mean SEM (Figure 5(b)) and 99%-SEM (Figure 5(d)). At load 5 requests per second, which corresponds to about 80% CPU utilization, Zeta' reduces average SEM by 28% and 99%-SEM by 80%[2]. This additional reduction at 99%-SEM comes from Zeta variance reduction.

5.4.3 Number of Unprocessed Requests

Another view that Zeta outperforms FifoP is the comparison on the percentage of unprocessed requests. As discussed at Section 5.1, a request is considered unprocessed if it does not get any processing time or if the processing time is too small. Figure 5(f) shows the percentage of unprocessed requests at Zeta is smaller than that of FifoP. This demonstrates the benefits of reservation: Zeta prevents earlier requests to starve later requests by reserving processing time for the later ones. Therefore, Zeta reduces the number of ”starved” requests.

6. Extended Performance Study

We use simulation to further evaluate the performance of Zeta-Online on aspects that cannot be conveniently evaluated through an implementation: (1) different quality profiles, and (2) a comparison of Zeta-Online to the offline optimal and to other online algorithms.



(a) (b) (c)

Figure 6. Simulation Results.

[2] The reduction for average SEM and 99%-SEM is computed using the target SEM value 0.05 as baseline. For example, the average SEM for FifoP is 0.064 with distance to base value 0.014; the average SEM for Zeta is 0.060 with distance to base value 0.01. So the average SEM reduction is $(0.014 - 0.01)/0.014 = 28\%$.

6.1 Sensitivity to Quality Profile

We evaluate Zeta-Online on concave quality profile and three additional quality profiles shown in Figure 6(a):

(1) **Setup profile.** The setup profile mimics the effect of a setup cost with non-productive processing equal to 20% of the service demand. The setup phase could, for example, be used to initialize data.

(2) **Staircase profile.** The staircase profile mimics a situation in which improvement in response quality is discrete.

(3) **Linear profile.** The linear profile mimics applications that perform a linear scan or iterative processing of random (unsorted) data that may not exhibit diminishing returns.

Simulation setup. Each request has a 100ms deadline interval, with service demand following Exponential distribution with 30ms mean. Request arrivals follow a Poisson distribution, and we vary load by changing mean arrival rate.

Results. Figure 6(b) shows that Zeta-Online outperforms FifoP for the setup and staircase profiles (which are not concave), and for the concave and linear profile (which are concave).

Next we discuss the quality loss of Zeta-Online under different quality profiles. Both the setup and staircase profiles are *concave at a coarse grain level*. The improvement in Zeta-Online is higher when the quality profile is more concave. Intuitively, with a more concave quality profile at the coarse grain level, the quality curve rises quickly at the beginning, and then exhibits diminishing returns, which is the property that Zeta-Online exploits: terminating a request that has been executing for a long time results in little quality loss for this request but saves processing time for other requests with potentially higher quality gain. Both setup and staircase are concave at a coarse grain, and therefore Zeta-Online provides better quality. The quality improvement for the linear profile is lower because it is *the least* concave function. In summary, Zeta-Online does not require the quality profile to be concave in a strict sense and provides better response quality under a variety of coarse-grain concave profiles.

6.2 Algorithm Comparison

This experiment compares Zeta-Online to offline optimal and to two well-known online algorithms from the prior work (MIG and FifoP): (1) Optimal: The optimal offline is an upper bound on the performance of any online algorithm. (2) MIG: Maximum Instantaneous Gain (MIG) is a well-known algorithm for QoS-adaptive system [2].[3] MIG optimizes for highest quality, running the ready request that gives the maximum instantaneous quality gain based on request quality profiles. Instantaneous gain is the increase in request quality given a unit increase in processing time. (3) FifoP: First-In-First-Out with partial results is the same algorithm used in previous sections, and it is equivalent to Earliest-Deadline-First (EDF) in our environment. It is well known that EDF is optimal in terms of meeting request deadlines when requests do not have partial execution.

Simulation setup. This experiment uses the same arrival and service distributions as the previous subsection along with the concave quality profile shown as "concave profile" in Figure 6(a). Figure 2 of Section 3.2.4 uses the same setup.

Results. Figure 6(c) compares the quality loss of the four algorithms. As expected, offline optimal has the best performance. Offline optimal performs better than Zeta-Online because it has exact information on request service demand and quality profile as well as future arrivals. Such information typically is unavailable to online schedulers. Zeta-Online has the second best performance, giving quality loss lower than the other online algorithms. *The results show the importance of considering both deadline and quality profiles*:

- FifoP is deadline aware: since the requests have the same deadline, FifoP executes the job with earliest deadline first. However, FifoP does not exploit the concavity of the quality profile so it underperforms compared to Zeta-Online.
- MIG optimizes for quality profile but it does not consider request deadline, resulting in suboptimal decisions. For example, suppose two requests have the same service demand of 60ms, arrival time at 0 and 50ms, and expiration time at 100 and 150ms respectively. MIG preempts the first job at 50ms to run the second job,

[3] QoS-adaptive system is described at related work (Section 7)

which is a bad decision because the second job has further deadline and can be executed after the first job. This drawback in MIG is more significant at light loads because better scheduling is likely to complete all requests. This explains why MIG has lower quality than FifoP at light load. Zeta-Online is consistently better than MIG, which demonstrates the importance of considering both deadlines and quality profiles.

7. Related Work

This paper presents Zeta scheduling framework for interactive applications supporting partial execution. In contrast to Zeta, most prior work on scheduling [14, 20, 23–27] does not consider partial execution; they use admission control and other scheduling techniques to manage server resources. They are applicable to different application models and are rather complementary to our work, and we do not discuss them further in this section. We focus on systems supporting partial execution.

7.1 Partial execution

Employing partial execution and approximated computation is an active area of research. partial execution has been used by anytime algorithms in AI [29]. Unlike most algorithms that run to completion, anytime algorithms provide a single answer after performing some fixed amount of computation. Loop perforation [18] offers compiler and runtime support for partial execution and has been applied to audio/video codecs. Web content adaptation [8, 15] offers different versions of contents for the same request. Baek and Chilimbi [3] presented a general framework to support approximated computation of different applications to tradeoff between quality and energy consumption.

The above prior work [3, 8, 15, 18, 29] offers important insights on how to adapt execution for different applications. They focus on partial execution mechanism that enables individual requests to produce partial results. They do not consider server environments where multiple requests are competing for resources and do not use the quality profile to allocate resources among requests for higher server capacity. They construct an important foundation for our work by enabling individual requests to support partial execution. We study how to schedule these adaptive requests in a server environment for better overall quality.

7.2 Server Systems

Zeta makes scheduling decision based on request quality profile and deadline. Among the prior work that considers partial execution, QoS-adaptive systems [2, 19, 28] apply utility profile, and soft real-time schedulers [7, 9, 10] perform deadline-oriented scheduling.

QoS-adaptive systems: These systems [2, 19, 28] map services with utility profiles to a pool of resources to maximize the aggregated quality. To support partial execution, each service can export multiple QoS levels of different quality and utility. For example, a network-intensive service can have its utility profile defined as its service qualities corresponding to different rates of allocated data bandwidth. QoS-adaptive systems look similar with Zeta, however, they are very different: QoS-adaptive system is stateless: at any time, allocating to the service the same amount of resource (such as data bandwidth) always gives the same quality gain regardless of the prior execution of the service. However, Zeta system is stateful: quality gain of a request is related to its prior processing time. Utility-based systems do not consider deadline either. Therefore, to optimize a utility-based system, greedily optimizing quality gain at each time unit gives a global optimal solution on the aggregated quality. However, it is no longer true for Zeta because an optimal local decision may result in suboptimal behavior for later execution, as we see from the performance results of MIG in Section 6.2. Adding time dimension into scheduling makes it a harder problem to solve.

Soft real-time systems: There is a large body of work on soft real-time scheduling [4, 5, 20, 23–25] to improve total quality (or rewards) subject to request deadline. Some of them [4] combine utility with deadlines. However, except the work we discuss below, they do not consider partial execution and therefore not applicable in our environment. The closest work to ours [7, 9, 10, 22] study adaptive (or partial) execution for multimedia applications. They consider weighted linear profile: executing a request with weight v_i for p_i amount of time, where $p_i \leq w_i$, gains quality $p_i v_i$. Another prior work [17] considers concave quality profiles; however, none of them [7, 9, 10, 17, 22] measures the quality profile of real-world applications to identify their concavity. Moreover, they don't report any implementation results.

Other systems: Partial execution is also used at web servers to improve overload behavior [1]. In particular, web server returns full contents at regular load and returns degraded-quality contents during overloading. This work offers an effective mechanism for overload handling but Zeta is designed for quality improvement for both regular and heavy load. Moreover, the work neither considers request quality profile nor offers a deadline guarantee for responses.

8. Conclusions

This paper introduces the Zeta scheduling model to improve scheduling for a common class of server workloads that possess three characteristics — partial execution, concave quality profile and time constraints. We propose an efficient online scheduler Zeta-Online that exploits the concavity of quality profile and deadline constraints to increase average response quality and high percentile quality without incurring preemption overhead and without needing exact job information on service demand. We implement and evaluate Zeta in Bing, in a finance server and by simulation. The results show that Zeta provides substantial benefits improving both average response quality and high-percentile quality.

Acknowledgments

We are grateful to Burton Smith, Chad Walters, Hongyang Sun, Junhua Wang, Kathryn McKinley, and Max Verun for their help and feedback.

References

[1] T. Abdelzaher and N. Bhatti. Web content adaptation to improve server overload behavior. In *WWW*, 1999.

[2] T. Abdelzaher, K. G. Shin, and N. Bhatti. User-level qos-adaptive resource management in server end-systems. *IEEE Trans. on Computers*, 52:2003, 2003.

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

[4] U. Balli, H. Wu, B. Ravindran, J. S. Anderson, and E. Douglas Jensen. Utility accrual real-time scheduling under variable cost functions. *IEEE Trans. Comput.*, 56(3):385–401, Mar. 2007.

[5] A. Block, B. Brandenburg, J. H. Anderson, and S. Quint. An adaptive framework for multiprocessor real-time system. In *Euromicro Conference on Real-Time Systems*, 2008.

[6] M. Broadie and P. Glasserman. Estimating security price derivatives using simulation. *Manage. Sci.*, 42:269–285, February 1996.

[7] E.-C. Chang and C.-K. Yap. Competitive online scheduling with level of service. In *Computing and Combinatorics*, 2001.

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

[9] F. Y. L. Chin and S. P. Y. Fung. Online scheduling with partial job values: Does timesharing or randomization help? *Algorithmica*, 37:149–164, 2003.

[10] M. Chrobak, L. Epstein, J. Noga, J. Sgall, R. van Stee, T. Tichý, and N. Vakhania. Preemptive scheduling in overloaded systems. *Journal of Computer and System Sciences*, 67:183–197, 2003.

[11] G. Cortazar, M. Gravet, and J. Urzua. The valuation of multidimensional american real options using the lsm simulation method. *Computers and Operations Research.*, 2006.

[12] B. Croft, D. Metzler, and T. Strohman. *Search Engines: Information Retrieval in Practice*. Addison Wesley, 2009.

[13] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. In *SOSP*, 2007.

[14] S. Elnikety, J. Tracey, E. Nahum, and W. Zwaenepoel. A method for transparent admission control and request scheduling in e-commerce web sites. In *WWW*, 2004.

[15] 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*, 5:10–19, 1998.

[16] J. Hamilton. Blog article: Perspectives. http://perspectives.mvdirona.com/2009/10/31/thecostoflatency.aspx. 2009.

[17] Y. He, S. Elnikety, and H. Sun. Tians scheduling: Using partial processing in best-effort applications. In *ICDCS*, 2011.

[18] H. Hoffmann, S. Sidiroglou, M. Carbin, S. Misailovic, A. Agarwal, and M. C. Rinard. Dynamic knobs for responsive power-aware computing. In *ASPLOS*, 2001.

[19] J. Huang, P.-J. Wan, and D.-Z. Du. Criticality- and qos-based multiresource negotiation and adaptation. *RTS*, 15: 249–273, 1998.

[20] C.-Y. Koo, T. W. Lam, T.-W. Ngan, and K.-K. To. On-line scheduling with tight deadlines. In *MFCS*, 2001.

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

[22] J. W. S. Liu, K.-J. Lin, W.-K. Shih, A. C.-s. Yu, J.-Y. Chung, and W. Zhao. Algorithms for scheduling imprecise computations. *Computer*, 24:58–68, 1991.

[23] C. Lu, J. A. Stankovic, T. Abdelzaher, S. H. Son, and G. Tao. Feedback control real-time scheduling: support for performance guarantees in unpredictable environments. *Oper. Syst. Rev.*, 34:33–43, 2000.

[24] A. Povzner, T. Kaldewey, S. Brandt, R. Golding, T. M. Wong, and C. Maltzahn. Efficient guaranteed disk request scheduling with fahrrad. In *Eurosys*, 2008.

[25] J. Stankovic, M. Spuri, K. Ramamritham, and G. C. Buttazzo. *Deadline Scheduling for Real-Time systems - EDF and related algorithms*. Academic Publishers, 1998.

[26] M. Welsh and D. Culler. Adaptive overload control for busy internet servers. In *USITS*, 2003.

[27] M. Welsh, D. Culler, and E. Brewer. Seda: an architecture for well-conditioned, scalable internet services. In *SOSP*, 2001.

[28] F. Yu, Q. Zhang, W. Zhu, and Y.-Q. Zhang. Qos-adaptive proxy caching for multimedia streaming over the internet. In *IEEE Trans. Circuits Syst. Video Techno.*, pages 257–269, 2002.

[29] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3), 1996.