Tians Scheduling: Using Partial Processing in Best-Effort Applications

Yuxiong He, Sameh Elnikety
Microsoft Research

Hongyang Sun
Nanyang Technological University

***Abstract*—To service requests with high quality, interactive services such as web search, on-demand video and online gaming keep average server utilization low. As servers become busy, queuing delays increase, and requests miss their deadlines, resulting in degraded quality of service with poor user experience and potential revenue loss. In this paper, we propose Tians scheduling, a group of scheduling algorithms for interactive services that can produce partial answers during overload. A Tians scheduler allocates processing time to each request based on system load with the objective of maximizing overall quality of responses.**

We propose three Tians scheduling algorithms — offline, online clairvoyant and online nonclairvoyant. For interactive applications with concave quality profile, we prove that the offline algorithm is optimal. We show the effectiveness of the online algorithms by conducting a simulation study modeling important applications — a web search engine and video-on-demand (VOD) system. Simulation results show a significant improvement of Tians over traditional server models: average response quality improves and the variance of responses decreases.

***Keywords*-interactive services, best-effort applications, offline, online clairvoyant, online nonclairvoyant, partial results, quality profile, scheduling, VOD bandwidth allocation, web search engine.**

I. Introduction

Many popular web applications are interactive, such as web search, map service, online gaming, and video-on-demand. Today, they constitute an important fraction of data center workloads. Service providers want to offer these services with high quality and short response time while reducing operational costs. To reduce cost, it is desirable to operate servers with high utilization, rather than using many lightly-loaded servers handling the same workload. Using fewer servers saves hardware, energy and maintenance costs. This is particularly challenging for interactive services: for batch workloads, server utilization can be increased close to 100%; however, for interactive applications, server utilization is kept low.

An important reason behind keeping server utilization low is that the traditional software framework cannot support a good quality of service for interactive applications when servers are heavily loaded. A server component is typically written to execute a request fully, or reject the request under overload. If a request is not fully processed by its deadline, it returns no answer on time, which results in wasted processing time and degraded service quality. The quality profile of a request is a step function, which requires a request to be processed to its full service demand to return the response. Due to stochastic nature of arrival and service distributions, this strict software model is unable to offer high service quality on well-loaded servers.

Let's use a simple example from classic queuing theory to illustrate the problem of a strict software model. We model a simple server as an M/M/1 queue with First-In-First-Out (FIFO) scheduling. Suppose the server needs to answer each request with deadline 100 ms and the mean service demand of requests is 15 ms. Each request returns a result with quality 1 if its full service demand is satisfied within the deadline, and 0 otherwise. We measure the overall service quality as the average quality of responses, which is equal to the probability that a response is computed within the deadline. To offer a service quality of 0.99, what is the highest request arrival rate this server can sustain? We use queuing theory to answer this question: W denotes latency of a request, λ denotes arrival rate, μ denotes service rate, and Q denotes service quality. We have $Q = P(W \leq 100) = 1 - e^{-(\mu-\lambda)100}$. To satisfy quality of 0.99, the arrival rate is at most 0.3 times of the service rate, which means that the server utilization is at most 30%. The question we address in this paper is how to offer the same quality of service with higher arrival rate and server utilization using partial results.

The intuition behind our solution comes from the following observation: many interactive applications find the best available result within a predefined response time. For example, in web search engine, such as Bing or Google, user query should be satisfied in a few hundred milliseconds, and similarly for other web services such as a map service, online travel arrangement, and online gaming. The returned response is usually not the only correct response in the entire search space; it is, however, the best available response given the response time limit. In other words, processing a request partially carries a meaningful value. Unlike the traditional server model whose quality follows a step function of processing time, such best-effort applications have intrinsic quality profile in which response quality improves with received processing time.

To take advantage of the quality profile in the interactive applications, we introduce Tians scheduling, which adopts partial results and maximizes response quality. More precisely, when a server software applies Tians scheduling, it accepts an allocated processing time as an input parameter with each request, and generates the best partial (or full) result given the amount of processing time. A Tians scheduler assigns processing time to each request using request characteristics and system load. Its goal is to maximize the overall quality of the responses subject to the deadline of requests. We develop three Tians scheduling algorithms: offline (known future arrivals and service demands), online

clairvoyant (known service demands for arrived requests) and online nonclairvoyant (unknown service demands).

We employ a simulation study to evaluate the benefits of Tians scheduling in two application domains: scheduling CPU processing times in a web search engine and scheduling upstream bandwidth in a video-on-demand environment. We find that Tians scheduling (a) improves average response quality and (b) reduces the variance of response quality. In particular, to offer the same quality of service, in the web search engine, Tians sustains more than 400% higher request arrival rate than a traditional FIFO server; for the VOD server, Tians sustains more than 40% higher load than FIFO.

The contributions of the paper are the following: (1) We propose Tians scheduling for best-effort interactive services, exploiting partial results to improve average response quality. (2) We propose three Tians scheduling algorithms for three important settings: offline, online clairvoyant, online non-clairvoyant. (3) We prove the optimality of the Tians offline scheduling algorithm when quality profile is concave. (4) We conduct a simulation study to assess the benefits of Tians online scheduling algorithms in two domains — a web search engine, and a video-on-demand (VOD) environment.

This paper is structured as follows. Section II describes partial results in server software. Section III gives an overview of Tians scheduling. Section IV presents an optimal offline algorithm, and proves algorithm optimality. Section V presents two online algorithms. Section VI evaluates our online algorithms through a simulation study. Section VII contrasts Tians scheduling to related work, and Section VIII concludes the paper.

II. Partial Results

This section explores the features of quality profiles in best-effort interactive applications and discusses how to support partial results.

Many best-effort applications such as web search, video streaming, online gaming, etc., have several responses for a request. They implicitly use a response quality function, in which response quality improves with request processing time. For example, in web search, when a user submits a search request for certain keywords, the search engine scans a distributed inverted index looking for webpages that match the keywords, and ranks the matching webpages. An inverted index lists important (e.g., popular) web pages first; therefore webpages searched earlier are more likely to rank higher and contribute more to total quality of the request. If the search engine does not finish identifying and ranking all matching webpages, it can still return the best matches found so far. Here, the quality of the response improves with processing time, and exhibits diminishing returns.

Response quality profiles are likely to be concave or close to concave because of the effect of diminishing returns and the iterative nature of the best-effort algorithms. A function f is *concave* if $f(\alpha x + (1-\alpha)y) \geq \alpha f(x) + (1-\alpha)f(y)$ for any $0 < \alpha < 1$ and valid input x and y. This function is *strictly concave* if the inequality always holds, i.e., $f(\alpha x + (1-\alpha)y) > \alpha f(x) + (1-\alpha)f(y)$. For example, Figure 1(a) shows a quality profile example used in search engine simulation. This profile is monotonically increasing and concave. The phenomena of diminishing returns appear in many domains. For example, in video streaming, basic layers are much more important than refinements layers; the quality of received video streams improves monotonically with the number of received layers but exhibits diminishing returns. In this paper, we exploit the properties of concavity in our scheduling algorithms.

In best-effort applications, there are various techniques to support partial results. Such techniques are beyond the scope of this paper but we point out that in many domains, a response is computed iteratively and therefore controlling the number iterations is an attractive technique. Servers would accept an additional parameter with each request specifying the allowed processing time. Such a parameter can be used to select the appropriate algorithm or dataset to compute or approximate the response.

III. Overview of Tians Scheduling

Compared to a traditional scheduling system, Tians scheduling has additional responsibility of allocating the processing time to each job (or request) based on system load, and job deadline.

Motivating Example: Consider this simple scenario, two jobs arrive at the system simultaneously, each with processing requirement of 100 ms. The deadline for each job is also 100 ms. Using FIFO, the first job meets the deadline while the second one misses it. Suppose that the job quality follows the quality profile in Figure 1(a). FIFO gets total quality 1. Due to the concavity of the quality profile, we can do better by partially executing the first job for 50 ms, and allocating the remaining time to the second one, resulting in 0.96 quality for each job and total quality 1.92. This example illustrates the benefit of an enhanced scheduler for maximizing overall quality of responses.

Scheduling Model: We use the following scheduling model. There is set $\mathcal{J} = \{J_1, J_2, \ldots, J_n\}$ of n jobs. Each job $J_i \in \mathcal{J}$ is characterized by an *arrival time* r_i, *deadline* d_i, and *service demand* (aka *total work*) w_i. For each job J_i, a scheduling algorithm starts to execute the job at time s_i and completes it at c_i, where $s_i \geq r_i$ and $c_i \leq d_i$. Let p_i denote the *processing time* job J_i receives during $[s_i, c_i]$, and it may not be processed to its full service demand, that is, partial execution is allowed with $p_i \leq w_i$. A *quality function* $f : R \to R$ maps the processing time a job receives to a quality value gained by executing the job. Moreover, we assume that the same quality function f applies identically to all jobs in $\mathcal{J}$. Table 1 summarizes the main notations used in the paper.

The objective is to maximize the total quality gained by executing all jobs in the job set, i.e., $\sum_{i=1}^{n} f(p_i)$. Without loss of generality, we assume that the arrival times of the jobs satisfy $r_1 \leq r_2 \leq \cdots \leq r_n$. In addition, we assume that the deadlines of the jobs are *agreeable*, that is, a job that arrives later naturally has a later deadline, i.e., $d_1 \leq$

Table I
TERMINOLOGY AND NOTATION.

Terminology	Notation
job set	$\mathcal{J}$
num of jobs in $\mathcal{J}$	n
quality function	f
parameters for job J_k:	
arrival time	r_k
deadline	d_k
service demand	w_k
start time	s_k
completion time	c_k
processing time	p_k

$d_2 \leq \cdots \leq d_n$. Tians scheduling algorithms target quality functions that are monotonically increasing and concave as discussed in Section II.

IV. Offline Scheduling

This section presents an offline scheduling algorithm: Tians-Optimal. We show that Tians-Optimal is an optimal offline algorithm when the quality function f is monotonically increasing and strictly concave. The optimal solution does not depend on the shape of the profile. Moreover, the optimal scheduler is non-preemptive, executing jobs in their arrival order. Implementing such a non-preemptive scheduler in server software is simpler than preemptive ones.

The intuition behind Tians-Optimal is that when quality profile of an application is concave, an optimal algorithm would allocate to each job an equal amount of time subject to job's service demand unless the allocation violates job's deadline. We define an allocation policy — Service-time Oriented Equal Partitioning (SOEP) — to apply the equal allocation principle of this observation. Roughly speaking, an optimal schedule is a sequence of segments, where the jobs inside the segment are scheduled by SOEP and the boundary of segments is decided by job deadline constraints. Applying this principle, we use the following three steps in Tians-Optimal. First, it identifies independent blocks of adjacent jobs, such that each block can be scheduled independently of others. Second, inside each independent block, it searches for the busiest segment. Once the busiest segment is identified, we schedule its jobs using SOEP. Here SOEP always produces a feasible schedule. Third, we remove this scheduled segment from the containing independent block, producing two smaller independent blocks that are scheduled recursively using the second step. We elaborate the concepts and algorithm in the remainder of this section.

A. Definitions and Notations

Before presenting Tians-Optimal and its analysis, we start this section by formalizing some basic concepts about a job block. Two jobs are *adjacent* if there is no job arriving between them. A *block* of jobs, represented by $\mathcal{J}[i,j]$, is a sequence of adjacent jobs from J_i to J_j. The *span* of $\mathcal{J}[i,j]$ is the interval $[r_i, d_j]$, which is the maximum time interval we can use to execute $\mathcal{J}[i,j]$. An independent block is defined as follows:

Definition 1: Two adjacent jobs J_i and J_{i+1} are *related* if $r_{i+1} < d_i$. Otherwise, they are not related since job J_{i+1} arrives after job J_i expires. The block $\mathcal{J}[i,j]$ is defined as an *independent block* if the following three conditions hold:

1. J_k and J_{k+1} are related for all k where $i \leq k < j$,
2. $i = 1$ or J_{i-1} is not related to J_i,
3. $j = n$ or J_{j+1} is not related to J_j.

An independent block does not have any influence on other independent blocks. Therefore, we can identify them and compute optimal solution for each independent block separately. The identification of independent blocks takes only a linear scan of jobs. Thus, without loss of generality, we assume that the job set $\mathcal{J}$ consists of only one independent block, and our algorithm is applied to one independent block of jobs.

Next we present the idea of SOEP, which performs equal partitioning of total available processing time to jobs subject to job's service demand (and not the deadlines). SOEP does not consider deadline of individual job during allocation, therefore a SOEP schedule is not always feasible for any job block. However, when SOEP is feasible, it produces an optimal schedule for the block. As an important observation leading to Tians-Optimal, we formalize and prove it in Lemma 4.

Assume that all jobs are available to run in a given time interval I, SOEP allocates to each job an equal share of processing time unless the job's service demand is less. In other words, small jobs get what they demand, and large jobs get an equal share of processing time. For example, suppose the length of interval I is 10, and there are three jobs J_1, J_2, and J_3 with total demand 2, 7, and 10 respectively. SOEP allocates to these jobs 2, 4 and 4 as their assigned processing time. Here, a job J_i is *satisfied* if $p_i \geq w_i$; otherwise if $p_i < w_i$, the job is *deprived*. A formal definition of SOEP condition is:

Definition 2: A schedule for a job block $\mathcal{J}[i,j]$ on interval $I = [r_i, d_j]$ satisfies the *SOEP condition* if all the following three conditions hold:

- *Equal deprived condition*: All deprived jobs are assigned the same processing time $\bar{p}(\mathcal{J}[i,j])$, which we call the *mean deprived time*.
- *Smaller satisfied condition*: All satisfied jobs have their assigned processing time less than or equal to the mean deprived time.
- *Fully occupied condition*: If there exists any deprived job, the sum of processing time over all jobs is equal to the length $d_j - r_i$ of the block span I.

We define *SOEP segment* to represent a block of jobs where SOEP is feasible.

B. Algorithm

The major challenge of Tians-Optimal is to identify SOEP segments and split the total available processing time of the job set among them to maximize overall job quality. To achieve the goal, we identify the busiest segment of jobs and

assign the entire span interval (maximum available processing time considering deadline) to these jobs using SOEP. The intuition comes from concavity of quality profile: an optimal schedule wants to give jobs similar processing time whenever possible. Since jobs in a busier segment is likely to have smaller processing time because of the deadline constraint, the job segment in the busiest interval should take over its entire span to maximize the processing time of jobs inside the segment. After we make allocation for jobs at the busiest segment, we remove the segment from the containing block, producing two smaller independent blocks, and perform scheduling recursively in a divide-and-conquer fashion.

Algorithm 1 shows the pseduocode of Tians-Optimal, which is initially invoked with $start = 1$ and $end = n$ for a job set $\mathcal{J}$ consisting of n jobs. Tians-Optimal finds the busiest block using FindBusiestUnmarkedBlock, which return two cases — *all-satisfied* and *some-deprived*. In the all-satisfied case, all jobs can be satisfied. FindBusiestUnmarkedBlock marks all jobs and return the flag $found = false$. Tians-Optimal starts each job J_k at its earliest possible time and processes it to its full service demand. In some-deprived case, there exist jobs that cannot be satisfied and $found$ is set to true. FindBusiestUnmarkedBlock finds the busiest block containing deprived jobs, and Tians-Optimal schedules the jobs in the busiest unmarked block $\mathcal{J}[i,j]$ by applying SOEP condition on the entire span interval $[r_i, d_j]$. Here the busiest block is defined as the block with smallest u-mean value, where u-mean is defined as follows:

Definition 3: Given a job block $\mathcal{J}[i,j]$, the *unmarked mean* or *u-mean* of the block measures the average available processing time for unmarked jobs in the block, which is given by the available length of the interval after satisfying all marked jobs divided by the total number of unmarked jobs, i.e.,

$$\widetilde{p}(\mathcal{J}[i,j]) = \frac{d_j - r_i - \sum_{J_k \in \mathcal{M}(\mathcal{J}[i,j])} w_k}{|\mathcal{U}(\mathcal{J}[i,j])|}$$

where $\mathcal{M}(\mathcal{J}[i,j])$ and $\mathcal{U}(\mathcal{J}[i,j])$ denote the set of marked jobs and the set of unmarked jobs in block $\mathcal{J}[i,j]$, respectively.

To calculate u-mean of a block, satisfied jobs must be marked; in other words, any job whose work is less than or equal to this u-mean will be marked. FindBusiestUnmarkedBlock marks jobs and computes u-mean in a while loop. Due to the connection between marked jobs and u-mean, after marking some jobs the smallest u-mean may get increased in the next iteration of the while loop and hence more jobs can possibly be marked. When all jobs are marked, all jobs can be satisfied, and FindBusiestUnmarkedBlock returns false to indicate all-satisfied case to its caller; otherwise, FindBusiestUnmarkedBlock returns a busiest unmarked block $\mathcal{J}[i,j]$ with u-mean $\widetilde{p}(\mathcal{J}[i,j])$. Any job J_k whose work satisfies $w_k \le \widetilde{p}(\mathcal{J}[i,j])$ must be marked and thus satisfied by the algorithm; those jobs in $\mathcal{J}[i,j]$ that are not marked will be deprived and share the same processing time, which is equal to the u-mean of the block.

Algorithm 1 Tians-Optimal

Require: $start$, end
Ensure: start time s_k and complete time c_k for each job J_k, where $k \in [start, end]$.

if $end < start$ **then**
  return
($found$, i, j, $\widetilde{p}$) = FindBusiestUnmarkedBlock($start$, end)
if (!$found$) **then**
  // all jobs are marked and can be satisfied.
  for $k = start : end$ **do**
    $s_k = \max\{r_k, c_{k-1}\}$
    $c_k = s_k + w_k$
  return
else
  // schedule jobs in busiest unmarked block by SOEP.
  for $k = i : j$ **do**
    $s_k = \max\{r_k, c_{k-1}\}$
    $c_k = s_k + \min\{w_k, \widetilde{p}\}$
  // update other jobs' arrival time or deadline
  for ($start \le k \le i-1$ and $d_k > r_i$) **do**
    $d_k = r_i$
  for ($j+1 \le k \le end$ and $r_k < d_j$) **do**
    $r_k = d_j$
  // recursively schedule other jobs
  Tians-Optimal ($start$, $i-1$)
  Tians-Optimal ($j+1$, end)

Algorithm 2 FindBusiestUnmarkedBlock

Require: $start$, end
Ensure: The busiest unmarked block.

$numMarked = 0$
while ($numMarked < end - start + 1$) **do**
  // find block with smallest u-mean.
  $\widetilde{p}_{smallest} = \infty, i = j = start$
  for $k = start : end$ **do**
    for $s = k : end$ **do**
      compute u-mean $\widetilde{p}(\mathcal{J}[k,s])$ for block $\mathcal{J}[k,s]$.
      if $\widetilde{p}(\mathcal{J}[k,s]) < \widetilde{p}_{smallest}$ **then**
        $\widetilde{p}_{smallest} = \widetilde{p}(\mathcal{J}[k,s])$
        $i = k$, $j = s$
  // mark jobs whose work is less than $\widetilde{p}_{smallest}$
  $marked = false$
  for each unmarked job $J_k \in \mathcal{J}[start, end]$ **do**
    if $w_k \le \widetilde{p}_{smallest}$ **then**
      $marked = true$ and mark job J_k
      $numMarked$++
  if (!$marked$) **then**
    return $(true, i, j, \widetilde{p}_{smallest})$
return $(false, 0, 0, 0)$

The schedule produced for $\mathcal{J}[i,j]$ in $[r_i, d_j]$ satisfies the SOEP condition, where the mean deprived time is equal to the u-mean of the block. (The feasibility of the schedule will be proven in Section IV-C.)

The complexity of this offline algorithm is $O(n^4)$. Finding smallest u-mean takes $O(n^3)$ time because of the pair-wise comparison of the job blocks and a linear time search for the marked jobs to compute u-mean of each block. In the worst case, only 1 job is marked each time and therefore finding smallest u-mean needs to be invoked n time, resulting in the stated complexity.

C. Optimality Proof

Theorem 1 shows the optimality of Tians-Optimal.

Theorem 1: Given any job set $\mathcal{J}$ and a monotonically-increasing and strictly-concave quality profile f, Tians-Optimal produces an optimal schedule.

Proof sketch: The key ideas of the proof are as follows. We present their formal analysis one-by-one in the remaider of the section.

- It is sufficient for Tians-Optimal to consider non-preemptive FIFO (First-In First-Out) schedules, because for any feasible schedule, there exists a feasible non-preemptive schedule that executes all jobs in the FIFO manner with the same overall quality. (Lemma 2).
- If a feasible schedule of a job block satisfies SOEP condition, this schedule is optimal. (Lemma 4).
- The busiest unmarked block $\mathcal{J}[i,j]$ returned from FindBusiestUnmarkedBlock is a SOEP segment, thus SOEP produces a feasible schedule in $\mathcal{J}[i,j]$. (Claim 2).
- An optimal FIFO schedule must assign the entire span interval $[r_i, d_j]$ to the segment as Tians-Optimal does; and it would assign the jobs inside the segment using SOEP as Tians-Optimal does. (Claim 3).
- Since the busiest unmarked block divides the remaining jobs into two smaller blocks, we can apply FindBusiestUnmarkedBlock again. We apply mathematical induction to complete the proof.

□

As the first step of the analysis, we show that, although preemption is allowed, it is sufficient to consider non-preemptive schedules that execute jobs in the FIFO manner. These schedules are attractive in practice because of their simple implementation without preemption overhead.

Lemma 2: For any feasible schedule, there exists a feasible non-preemptive schedule that executes all jobs in the FIFO manner with the same overall quality.

Proof: Given a feasible 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 thus has early 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. Apparently, 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 is non-preemptive. ■

We know that the schedule produced by the Tians-Optimal algorithm is a non-preemptive FIFO schedule. We need to define more notations for the subsequent analysis.

Definition 4: A job J_h is said to be the *head* of job J_k in a schedule if h is the largest job index that satisfies the following conditions:

1. $1 \leq h \leq k$,
2. $s_h = r_h$,
3. $s_l = c_{l-1}$ for each $h < l \leq k$.

Similarly, a job J_t is the *tail* of job J_k in a schedule if t is the smallest job index that satisfies the following conditions:

1. $k \leq t \leq n$,
2. $c_t = d_t$,
3. $c_l = s_{l+1}$ for each $k \leq l < t$.

Moreover, the block $\mathcal{J}[h,t]$ is said to be the *tight block* of job J_k in this schedule.

Intuitively, the tight block of a job forms a continuous sequence of job execution containing the job. In a given schedule, not every job has the head or the tail, hence the tight block. However, in the following lemma, we show that such a tight block can always be found for deprived jobs in OPT, where OPT is an optimal FIFO schedule. We use p_k^* to denote the processing time job J_k receives in OPT.

Lemma 3: Any deprived job J_k in OPT belongs to a tight block $\mathcal{J}[h,t]$ and its processing time is not less than that of any other job in the block, i.e., $p_l^* \leq p_k^*$ for all $h \leq l \leq t$.

Proof: We first show that given a deprived job J_k in OPT, its head J_h must exist. Suppose that J_h does not exist, then there must be a continuous sequence of jobs in front of J_k with the leading job starting after its arrival. In this case, we can shift the starting and completion times of these jobs earlier by a small amount of time δ, which can then be assigned to J_k so that we gain more quality from it. This contradicts the fact that OPT is an optimal schedule. Therefore, J_h must exist. Now, suppose that there exists a job $J_l \in \mathcal{J}[h,k]$ whose processing time is more than that of J_k. In this case, we can reduce the processing time of J_l by a small amount of time δ, shift the starting and completion times of all jobs from J_l to J_k by δ, and then increase the processing time of J_k by δ. Since the quality function is strictly concave, the quality increase at J_k is larger than the quality decrease at J_k; the new schedule has higher quality than the original one. This again contradicts the fact the original schedule is optimal. Thus, we have $p_l^* \leq p_k^*$.

Similarly, we can show that the tail J_t of job J_k also exists and any job $J_l \in \mathcal{J}[k,t]$ satisfies $p_l^* \leq p_k^*$. ■

The following lemma shows the optimality of the SOEP condition, provided that it is feasible for a block of jobs.

Lemma 4: If SOEP is feasible for a job block $\mathcal{J}[i,j]$ in interval $I = [r_i, d_j]$, it is also an optimal schedule for $\mathcal{J}[i,j]$.

Proof: Let us consider a relaxed case, where all jobs in $\mathcal{J}[i,j]$ are eligible to run in the entire interval I. An optimal quality for this case is apparently an upper bound on the optimal quality for the original problem. For the relaxed problem, we will show that SOEP offers an optimal schedule. Therefore, when SOEP is also feasible for the original problem, it offers an optimal schedule as well.

For the relaxed problem, let $\mathcal{S}(\mathcal{J}[i,j])$ and $\mathcal{D}(\mathcal{J}[i,j])$ denote the set of satisfied jobs and the set of deprived jobs in SOEP. In addition, let $\bar{p}(\mathcal{J}[i,j])$ denote its mean deprived time. Now, consider any other fully occupied schedule χ that does not assign more processing time to a job than the job's total work. We will show how to transform χ to SOEP without decreasing the overall quality, thus proves that SOEP is optimal.

Suppose that in χ there exists a job $J_k \in \mathcal{S}(\mathcal{J}[i,j])$ that is deprived, so we have $p_k < \bar{p}(\mathcal{J}[i,j])$. Since χ is fully occupied and does not assign more time to a job than its work, the total processing time allocated to all jobs in $\mathcal{D}(\mathcal{J}[i,j])$ by χ is more than that by SOEP. Let J_l denote the job in $\mathcal{D}(\mathcal{J}[i,j])$ whose processing time is the largest in χ. Therefore, we should have $p_l > \bar{p}(\mathcal{J}[i,j]) > p_k$. Since the quality function is identical for all jobs and strictly concave, reducing a small amount of processing time δ for J_l and assigning it to J_k will lead to better overall quality. Repeating this process till all jobs in $\mathcal{S}(\mathcal{J}[i,j])$ are satisfied produces schedule χ'. To transform χ' to SOEP, repeatedly reassign a small amount δ from the job with the largest processing time to the job with the smallest processing time in $\mathcal{D}(\mathcal{J}[i,j])$ till all jobs in $\mathcal{D}(\mathcal{J}[i,j])$ have the same processing time. Again, the overall quality only increases due to the concavity of the quality function. ■

Now, we focus on Tians-Optimal and claim some useful properties. In particular, these properties apply to Tians-Optimal from the very beginning of its invocation to the time when it either finds the first busiest unmarked block or marks all jobs in the job set.

Claim 1: Any marked job is satisfied in OPT.

Proof: We prove the claim by contradiction. Suppose that there exists a marked job J_k that is deprived in OPT. According to Lemma 3, J_k belongs to a tight block $\mathcal{J}[h,t]$ in OPT and we have $p_l^* \le p_k^* \le w_k$ for all $h \le l \le t$. Moreover, since $\mathcal{J}[h,t]$ is a tight block and OPT will not assign more processing time to a job than its total work, we have $\sum_{l=h}^{t} p_l^* = d_t - r_h$ and $p_l^* \le w_l$ for $h \le l \le t$.

Let $\mathcal{M}(\mathcal{J}[h,t])$ and $\mathcal{U}(\mathcal{J}[h,t])$ denote the set of marked jobs and the set of unmarked jobs in block $\mathcal{J}[h,t]$ immediately before job J_k is marked by FindBusiestUnmarkedBlock. The u-mean of the block is then given by

$$\begin{aligned}\widetilde{p}(\mathcal{J}[h,t]) &= \frac{d_t - r_h - \sum_{J_l \in \mathcal{M}(\mathcal{J}[h,t])} w_l}{|\mathcal{U}(\mathcal{J}[h,t])|} \\ &\le \frac{d_t - r_h - \sum_{J_l \in \mathcal{M}(\mathcal{J}[h,t])} p_l^*}{|\mathcal{U}(\mathcal{J}[h,t])|} \\ &= \frac{\sum_{J_l \in \mathcal{U}(\mathcal{J}[h,t])} p_l^*}{|\mathcal{U}(\mathcal{J}[h,t])|} \le p_k^* \le w_k^* \ . \end{aligned} \quad (1)$$

When FindBusiestUnmarkedBlock decides whether J_k should be marked, it evaluates every block including $\mathcal{J}[h,t]$. According to Algorithm 2 and Inequality (1), J_k should not be marked. This contradicts the fact that it is marked. ■

Claim 2: The schedule produced by Tians-Optimal is feasible.

Proof: We will prove the feasibility by showing that the completion time assigned to each job by Tians-Optimal is no later than its deadline. We prove the claim by contradiction. Suppose that J_k is the first job violating its deadline, i.e., $c_k > d_k$. Since Tians-Optimal always starts the job at the earliest possible time decided by $\max\{c_{k-1}, r_k\}$, the head J_h of job J_k always exists. As mentioned previously, there are two cases depending on whether all jobs in the job set are marked or a busiest unmarked block is found. We consider these two cases separately in the following.

In the first case, where all jobs are marked, the processing time assigned to each job by Tians-Optimal is equal to its full service demand. Since we assumed $c_k > d_k$, according to Tians-Optimal, we have $d_k - r_h < \sum_{l=h}^{k} w_l$. Hence, the total work in block $\mathcal{J}[h,k]$ is more than the length of the span $[r_h, d_k]$ of the block. Thus, no schedule can possibly satisfy all jobs in the block. However, since all jobs are marked, according to Claim 1, they are all satisfied by OPT. This introduces contradiction. Therefore, the schedule produced by Tians-Optimal is feasible in this case.

In the second case, the busiest unmarked block is found. Let J_h denote the head of J_k. We have $h \ge i$ since otherwise J_i will be the head of job J_k. Consider the subblock $\mathcal{J}[h,k] \subseteq \mathcal{J}[i,j]$, and let $\mathcal{M}(\mathcal{J}[h,k])$ and $\mathcal{U}(\mathcal{J}[h,k])$ denote the set of marked jobs and the set of unmarked jobs in $\mathcal{J}[h,k]$. Since we assumed $c_k > d_k$, according to Tians-Optimal, we have $d_k - r_h < c_k - r_h = \sum_{J_l \in \mathcal{M}(\mathcal{J}[h,k])} w_l + |\mathcal{U}(\mathcal{J}[h,k])|\,\widetilde{p}(\mathcal{J}[i,j])$. The u-mean of block $\mathcal{J}[h,k]$ is then given by

$$\begin{aligned}\widetilde{p}(\mathcal{J}[h,k]) &= \frac{d_k - r_h - \sum_{J_l \in \mathcal{M}(\mathcal{J}[h,k])} w_l}{|\mathcal{U}(\mathcal{J}[h,k])|} \\ &< \frac{|\mathcal{U}(\mathcal{J}[h,k])|\,\widetilde{p}(\mathcal{J}[i,j])}{|\mathcal{U}(\mathcal{J}[h,k])|} = \widetilde{p}(\mathcal{J}[i,j]) \ . \end{aligned}$$

Since the u-mean of $\mathcal{J}[h,k]$ is less than that of $\mathcal{J}[i,j]$, it contradicts the fact that $\mathcal{J}[i,j]$ is the busiest unmarked interval. Thus, the schedule is also feasible in this case. ■

Claim 3: OPT assigns the same interval $[r_i, d_j]$ to the busiest unmarked block $\mathcal{J}[i,j]$ as Tians-Optimal does.

Proof: We again prove the claim by contradiction. Suppose that OPT assigns a sub-interval $[x,y] \subset [r_i, d_j]$ to the busiest unmarked block $\mathcal{J}[i,j]$ found by Tians-Optimal. Since the schedule produced by Tians-Optimal for $\mathcal{J}[i,j]$ satisfies the SOEP condition, and according to Claim 2 it is feasible, the interval $[r_i, d_j]$ is fully occupied. Moreover, all unmarked jobs in $\mathcal{J}[i,j]$ are deprived and they receive the same processing time equal to the u-mean $\widetilde{p}(\mathcal{J}[i,j])$. Since OPT assigns a sub-interval $[x,y]$ to $\mathcal{J}[i,j]$, there must be deprived jobs for $\mathcal{J}[i,j]$ in OPT, too. According to Claim 1, all marked jobs are satisfied by OPT. Hence, the total processing time given to the unmarked jobs by OPT is less than that by Tians-Optimal. Let J_k denote the unmarked job in $\mathcal{J}[i,j]$ whose processing time is the smallest given by OPT, and let $\mathcal{M}(\mathcal{J}[i,j])$ and $\mathcal{U}(\mathcal{J}[i,j])$ denote the set of

marked jobs and the set of unmarked jobs. We have

$$\begin{aligned} p_k^* &\leq \frac{y - x - \sum_{J_l \in \mathcal{M}(\mathcal{J}[i,j])} w_l}{|\mathcal{U}(\mathcal{J}[i,j])|} \\ &< \frac{d_j - r_i - \sum_{J_l \in \mathcal{M}(\mathcal{J}[i,j])} w_l}{|\mathcal{U}(\mathcal{J}[i,j])|} = \widetilde{p}(\mathcal{J}[i,j]) \ . \end{aligned}$$

Moreover, since J_k is unmarked by Tians-Optimal, we have $p_k = \widetilde{p}(\mathcal{J}[i,j]) \leq w_k$. We can conclude that $p_k^* < \widetilde{p}(\mathcal{J}[i,j]) \leq w_k$, and J_k is deprived in OPT.

Based on Lemma 3, job J_k belongs to a tight block $\mathcal{J}[h,t]$ in OPT, and clearly we have $\mathcal{J}[h,t] \neq \mathcal{J}[i,j]$ since otherwise it will be the case that $[x,y] = [r_i, d_j]$. Let $\mathcal{M}(\mathcal{J}[h,t])$ and $\mathcal{U}(\mathcal{J}[h,t])$ denote the set of marked jobs and the set of unmarked jobs in block $\mathcal{J}[h,t]$ immediately before $\mathcal{J}[i,j]$ is identified as the busiest unmarked block. Based on the same argument for proving Claim 1, in particular Inequality (1), we have that the u-mean of block $\mathcal{J}[h,t]$ satisfies $\widetilde{p}(\mathcal{J}[h,t]) \leq p_k^* < \widetilde{p}(\mathcal{J}[i,j])$. This contradicts the fact that $\mathcal{J}[i,j]$ is the busiest unmarked block. ■

With the help of these claims, we can now complete the optimality proof of Theorem 1.

Theorem 1: Given any job set $\mathcal{J}$ and a monotonically-increasing and strictly-concave quality profile f, Tians-Optimal produces an optimal schedule.

Proof: We consider two cases. In the first case, all jobs in job set $\mathcal{J}$ are marked. Based on Tians-Optimal, each job will receive its full service demand, and according to Claim 2, the schedule produced by Tians-Optimal is feasible. Hence, the overall quality is the maximum possible, so the schedule is optimal.

In the second case, a busiest unmarked blocks is found. We prove the optimality of Tians-Optimal in this case using mathematical induction with respect to the number of jobs in $\mathcal{J}$. When $\mathcal{J}$ contains only one job, Tians-Optimal clearly produces an optimal schedule. Hence, the base case is trivial. The following shows the inductive step.

Inductive hypothesis: When $|\mathcal{J}| < n$, Tians-Optimal produces an optimal schedule.

Inductive step: We will show that when $|\mathcal{J}| = n$, Tians-Optimal also produces an optimal schedule. Let $\mathcal{J}[i,j]$ denote the first busiest unmarked block found by Tians-Optimal. According to Claim 3, both Tians-Optimal and OPT will assign interval $[r_i, d_j]$ to this block. Moreover, the schedule for $\mathcal{J}[i,j]$ in $[r_i, d_j]$ also satisfies the SOEP condition and according to Claim 2, it is feasible. Therefore, based on Lemma 4, Tians-Optimal produces an optimal schedule for $\mathcal{J}[i,j]$. After that, Tians-Optimal divides the remaining jobs into two independent blocks $\mathcal{J}[1, i-1]$ and $\mathcal{J}[j+1, n]$, both of which according to the inductive hypothesis are scheduled optimally by Tians-Optimal. Therefore, Tians-Optimal produces an optimal schedule for the entire job set $\mathcal{J}$. ■

Although the analysis of Theorem 1 is based on strict concavity, it is not hard to see that Tians-Optimal offers an optimal solution for any concave quality profile. When the profile is concave but not strictly concave, the optimal solution may not be unique and Tians-Optimal finds one of them.

V. Online Scheduling

Online algorithms do not possess future information of the jobs, such as arrival and service demand information. So a scheduler makes decisions based on arrived jobs. There are two types of online schedulers — online clairvoyant and online nonclairvoyant. An online clairvoyant scheduler knows complete information of arrived jobs, which includes a job's arrival time, deadline and service demand in our model; in contrast, an online nonclairvoyant scheduler possesses information only on the executed part of the job, and it does not know the job service demand. We develop two online algorithms that employ Tians-Optimal to simplify the presentation.

A. Online Clairvoyant Algorithm

Tians-Clairvoyant applies Tians-Optimal to the set of ready jobs (jobs that have arrived and not expired). Tians-Clairvoyant runs Algorithm 3 to assign processing time to jobs. This algorithm can be applied in both preemptive and non-preemptive scenarios depending on server software implementation: (1) the processing time of a job is assigned once before execution and cannot be changed (no preemption); (2) the scheduler can change the assigned processing time of a job by interrupting the job at any time. For case (1), the algorithm is invoked when the system has ready jobs in the $queue$ with no active job. A job is executed until it completes or its assigned processing time elapses. For case (2), a newly arrived job may change the assigned processing time of the current active job. Therefore, the assigned processing time of the active job is recomputed with upon each new arrival. If the job's elapsed processing time is greater than or equal to its assigned processing time, the scheduler terminates the job immediately. Tians-Clairvoyant adjusts arrival time of jobs in the ready queue (Line 1 to 3) so that Tians-Optimal assigns processing time from current time onwards.

Algorithm 3 Tians-Clairvoyant (Jobs[] $queue$, Job $active$, double $curtime$)

Require: $active$ denotes the current running job;
$queue$ denotes list of ready jobs (including active job at $queue[0]$ if $active$ is not null);
$curtime$ represents current time stamp;

1: $timeBase = (active == null)?\ curtime : s_{active}$
2: **for** ($k \in queue$ && $r_k < timeBase$) **do**
3: $r_k = timeBase$
4: Tians-Optimal ($queue$)
5: $p_{queue[0]} = c_{queue[0]} - timeBase$
6: **return** $queue[0]$

B. Online Nonclairvoyant Algorithm

Since a nonclairvoyant scheduler cannot foresee the service demand of jobs, we optimize over expected service demand.

Tians-NonClairvoyant applies the Tians-Clairvoyant algorithm with jobs' expected (average) service demand. We replace the actual service demand of jobs in the queue with their average service demand (Line 2), and assume that the active job (or incoming active job) has its service demand equal to the total span (Line 3) rather than expected service demand for the two reasons: (1) When the system is lightly loaded, if we use service demand to decide processing time of an active job, the active job will get processing time no larger than its average service demand even when there is ample processing time. This raises a problem when the active job has service demand larger than the expected service demand; Therefore we always assume the active job is a large job to allow it to use additional processing time. When the active job turns out to be a small one, it completes before its assigned processing time and scheduler can carry on the remaining jobs. (2) When the system is heavily loaded, a good decision is to apply something like equal partitioning. In this case even when we assume the active job is large, the algorithm still tries to allocate each job an equal amount of processing time as long as it does not violate jobs' deadline.

Algorithm 4 Tians-NonClairvoyant (Jobs[] $queue$, Job $active$, double $curtime$)

Require: $active$ denotes the current running job;
$queue$ denotes list of ready jobs (including active job at $queue[0]$ if $active$ is not null);
$curtime$ represents current time stamp;

1: **for** ($k \in queue$ && $k != queue[0]$) **do**
2: w_k = average service demand of jobs
3: $w_{queue[0]} = span_{queue[0]}$
4: **return** Tians-Clairvoyant ($queue$, $active$, $curtime$)

In practice, the implementation of both algorithms can be improved to obtain better time complexity: Given m ready jobs, the complexity of Tians-Optimal to compute assigned processing time is $O(m^4)$. Since we consider online scheduling scenarios in which a scheduler only knows the jobs that already arrived, the busyness of an interval can only degrade over time. Therefore, the busiest block starts from only the current time, which reduces the triple loop inside FindBusiestUnmarkedBlock (Algorithm 2) to double loop. So the complexity of Tians-Clairvoyant and Tians-NonClairvoyant is reduced to $O(m^3)$. In addition, Tians-Clairvoyant needs to compute the assigned processing time for jobs only when a new job arrives; during job departure, it simply uses the assigned processing time computed when last job arrived.

VI. Evaluation

We conduct a simulation study to assess the benefits of Tians scheduling. We evaluate it using two applications — web search engine, modeling a large commercial web search engine such as Bing, and VOD, modeling a video-on-demand system. For the search engine, to schedule CPU processing time, the service time of a request is not known upon its arrival, thus we apply Tians-NonClairvoyant; in contrast, for VOD, to schedule network bandwidth, data size of a request is known upon its arrival, thus we apply Tians-Clairvoyant. We find that for the search engine, to offer service quality 0.99, Tians-NonClairvoyant sustains more than 400% increase in request arrival rate; for VOD server, to offer service quality 0.9, Tians-Clairvoyant sustains more than 40% increase in the number of supported clients.

We model server applications using a combination of discrete-event simulation and queuing models. Upon the arrival or departure of a request, the server scheduler performs regular enqueue and dequeue operations, and decides if the new status of the queue changes the assigned processing time of the running request. The quality of a request is calculated based on its processing time and the quality profile of the requests.

A. Scheduling Algorithms

We implement six algorithms: three Tians algorithms (Tians-NonClairvoyant, Tians-Clairvoyant, and Tians-Optimal), as well as three algorithms based on FIFO:

FIFO-NonClairovyant simulates a nonclairovoyant traditional system with a FIFO queue without support for partial results. The system does not know request service time and discards requests that have already expired before the processing starts. However, once it is started a request always runs through completion. If a request i is answered past its deadline, the result quality is 0, otherwise, the request receives its full quality, i.e., $q_i = f(w_i)$.

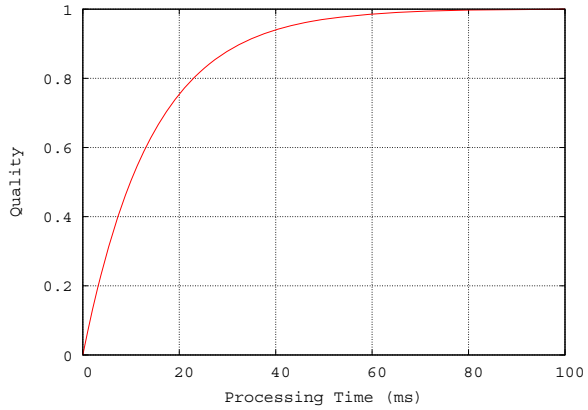
FIFO-Clairvoyant simulates a clairovoyant traditional system with a FIFO queue without support for partial results. The system knows request service time and discards a request if the request cannot be completed before its deadline. A discarded request has quality 0, and a completed request receives its full quality, i.e., $q_i = f(w_i)$.

FIFO-Partial uses partial processing from Tians, and uses FIFO to schedule requests. A request is discarded if it is already expired before processing starts. Request processing stops upon request deadline or request completion. The quality of a request i depends on the actual processing time it receives, i.e., $q_i = f(p_i)$.

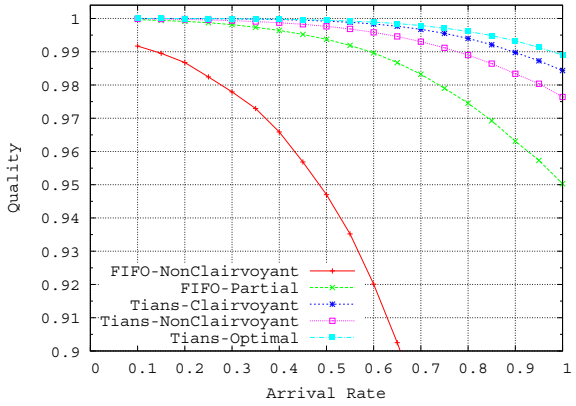
B. Web Search Engine

1) Description: The simulation models a web search engine back-end, which accepts user queries, searches its index for all documents that match the query, and ranks these documents before returning the top N documents that match the query in the rank order. Web search is a best-effort interactive application where the result quality improves with the increased number of documents examined and ranked, and the results need to be returned in a given amount of time.

2) Simulation Setup: Our experiments use the following parameters: request deadline 150 ms, and target request quality 0.99. Service demand follows an exponential distribution with mean service demand of 26 ms but the distribution is capped at 150 ms. Requests arrive according



(a) Quality profile



(b) Performance results

Figure 1. Web search engine (a) quality profile and (b) performance results comparison with varying normalized arrival rate.

to a Poisson process, and we vary the mean request arrival rate, as a ratio of the service rate to control system load.

The search engine experiments are performed with a quality profile represented by a concave quality function as follows,

$$f(x) = \frac{1 - e^{-cx}}{1 - e^{-100c}} \ ,$$

where $c = 0.07$ is a multiplier constant, and x is the actual processing time of a request. The quality profile is drawn in Figure 1(a), where the x-axis represents the processing time of a request and the y-axis represents the quality (relevance) of the returned results.

3) Main result: One major benefit of using Tians scheduling is offering the same quality of service with higher request arrival rate. Figure 1(b) depicts quality (y-axis) against normalized arrival rate (x-axis), which is the arrival rate normalized to service demand, representing system load. At quality 0.99, FIFO-NonClairvoyant supports an arrival rate of 0.15, while Tians-NonClairvoyant can sustain an arrival rate of 0.78. Therefore, Figure 1(b) shows that Tians-NonClairvoyant increases the supported arrival rate by more than 400% of FIFO-NonClairvoyant. From another point of view, we relate quality with system utilization. To achieve 0.99 quality, FIFO-NonClairvoyant allows server utilization around 15% while Tians-NonClairvoyant supports server utilization around 78%. Because Tians supports higher arrival rate and higher server utilization, it can be exploited to deploy a smaller number of servers to process the same workload. At quality 0.99, to support the same total arrival rate of requests, Tians-NonClairvoyant requires less than $1/5$ of the servers required in FIFO-NonClairvoyant.

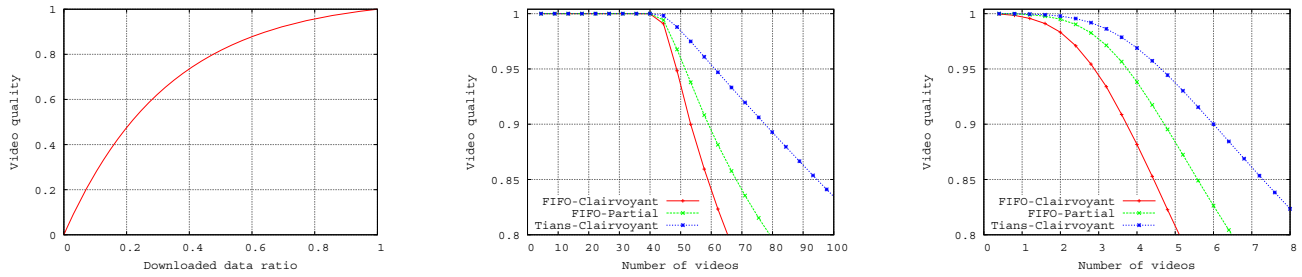
4) Discussion: To illustrate the factors contributing to the performance gain, Figure 1(b) presents the results using FIFO-Partial, Tians-Clairvoyant and Tians-Optimal.[1] At quality 0.99, FIFO-NonClairvoyant supports an arrival rate of 0.15, FIFO-Partial 0.60, Tians-NonClairvoyant 0.78, Tians-Clairvoyant 0.90, and Tians-Optimal 0.97. To evaluate the benefit of partial execution, we compare FIFO-NonClairvoyant with FIFO-Partial: partial results increase the arrival rate by 300% (contribution of partial evaluation). To show the importance of scheduling, we compare FIFO-Partial with Tians-NonClairvoyant: the enhanced scheduling algorithm further improves the arrival rate by 120% (contribution of better scheduling). If predicting service demand of requests is possible, we can use Tians-Clairvoyant to improve the arrival rate further with another 80% (contribution of knowing service demand). Knowing future arrivals, allows further improvement of 47%.

Tians-Clairvoyant performs close to the offline optimal, Tians-Optimal, which can sustain arrival rate of 0.97 with the full knowledge of future arrivals and service demands.

[1]Tians-Clairvoyant and Tians-Optimal use additional information such as query service demand and future arrival that are not available during online processing of the web search engine. We present their results for reference purpose.

C. VOD Bandwidth Allocation

1) Model: A video-on-demand (VOD) server is a complex distributed system that receives requests for continuous media (e.g., video files) from clients and streams the requested media back. A VOD server manages many resources such as network upstream bandwidth and disk I/O. One of the fundamental problems that a VOD system faces is admission control in which a request for a new media stream is either granted (with an implicit promise for a certain quality of service) or declined (at a potential revenue loss). Here we apply Tians scheduling to manage the upstream bandwidth. We assume that the server has an upstream network link with a limited bandwidth to send media packets to clients. We model a modern VOD configuration in which the data are encoded using a scalable streaming framework [14] with quality adaptation, where the quality of the video improves with the amount of data received from the server before the playback deadline [23]. Table II summarizes the model parameters. We use the quality profile depicted in Figure 2(a) (exponential quality function) to represent the quality of video as a function of the received data ratio (actual data received normalized to the full service demand), which exhibits diminishing



(a) Quality profile (b) VOD Server (c) Collaborative VOD environment

Figure 2. Quality profile and key results of VOD: : (a) quality profile of VOD application; (b) quality of the video streams at VOD server with varying the number of supported clients. (c) quality of the video streams at a collaborative environment with varying the number of supported clients.

Table II
MODEL PARAMETERS FOR VOD SERVER

Parameter	Value
Upstream bandwidth	1Gbps
Video rate	uniform (5Mpbs, 40Mpbs)
Deadline	2secs
Video quanta	2secs · video rate

returns.[2]

The input to the scheduler is a sequence of requests for video quanta with their sizes and deadlines. The scheduler assigns to each request a permitted size of data to send on the upstream link. If the system is under-loaded, most requests are granted. As the system is under high load and the upstream bandwidth becomes insufficient to satisfy all requests fully, the scheduler may allocate less data to send than demanded on the upstream link. The objective of the scheduler is to allocate the upstream bandwidth among requests to maximize the overall system quality. This model uses a clairvoyant scheduler since the data size of a video quantum is known: the information of data size can be obtained by adding metadata to the video stream.

2) Main result: Figure 2(b) shows the average video stream quality against system load. The x-axis is the number of clients, each viewing a single video. The y-axis is the average quality of received video streams. The figure has two curves for FIFO-Clairvoyant and Tians-Clairvoyant. At light load, the two curves are identical because bandwidth is an abundant resource. As the VOD server streams more video clips (at 40 Clients), scheduling improves the supported video stream quality; in particular, at 90% quality Tians supports an additional 23 video streams.

3) Collaborative VOD environment: Next, we apply Tians scheduling to a collaborative VOD environment in which a client may send video streams to other clients as proposed in prior work [21], [24]. Clients use Tians to schedule requests on their upstream links. We assume that the bandwidth of a client upstream is 1 Mpbs and video streams have variable bit rates following a uniform distribution from 100 Kbps to 400 Kbps. A request asks for a video quantum of 2 seconds.

Figure 2(c) shows the effect of Tians clairvoyant scheduling. There are noticeable performance differences between FIFO-Clairvoyant and Tians-Clairvoyant at low loads since the upstream bandwidth is a scare resource. We also add FIFO-Partial which schedules requests in FIFO order with credit for partial data. At quality 0.9, from FIFO-Clairvoyant to FIFO-Partial, partial results increase the number of supported videos from 3.6 to 4.6, representing the contribution of using partial results. From FIFO-Partial to Tians-Clairvoyant, the enhanced scheduling algorithm further improves the number of supported videos from 4.6 to 6, representing the contribution of better scheduling.

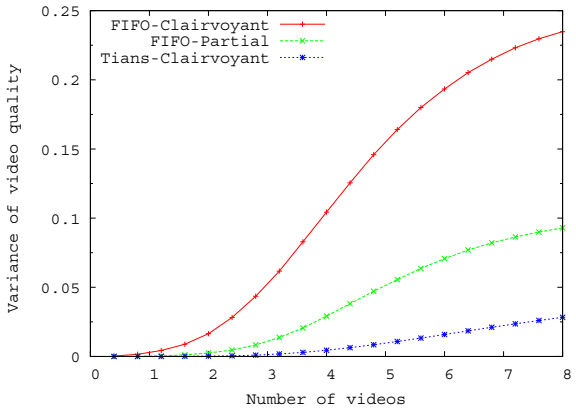


Figure 3. Variance of video stream quality in a collaborative environment with varying the number of supported clients. Its corresponding average quality result is presented in Figure 2(c).

D. Variance Study

Tians reduces variance of response quality in addition to improving the average quality. Small variance in a distributed environment is desirable because it helps to improve service quality which often uses a high-percentile quality as a performance metric in addition to average quality. Figure 3 shows that variance of video stream quality at collaborative environment produced by Tians-Clairvoyant is much smaller than that of FIFO-Clairovyant. Two factors

[2]When the quality profile takes normalized processing time instead of absolute processing time as input, the optimal solution depends on the shape of the profile and Tians-Optimal may not be optimal. However, experiments in this section show that online Tians model still outperforms traditional significantly.

contribute to variance reduction of Tians. (1) Smooth quality profile with partial results: if we model the quality of a response using a random variable $X \in [0, 1]$, at a traditional server, X follows Bernoulli distribution and, at any given mean $E[X] = \mu$, it produces maximum possible variance $\delta^2 = \mu - \mu^2$; Tians uses smooth quality profile to reduce the variance of the response quality. (2) Equal-partitioning at scheduling: Tians tries to give each request an equal amount of processing time whenever possible, which reduces the processing time differences among requests and contributes to reduced variance.

VII. Related Work

In real-time systems with deadlines, Earliest Deadline First (EDF) algorithm gives an optimal solution under light load, i.e., when the deadlines of all jobs can be satisfied. Otherwise, EDF does not perform well. Hence, different system and scheduling models have been proposed.

Strict scheduling (no partial evaluation): Many early results [5], [15]–[19] use a strict scheduling model where a job has work w_i and a weight v_i. If the job is completed before its deadline, it gains a quality $w_i v_i$; otherwise, it does not gain any quality. The objective is to maximize the overall quality gain. This model is similar to the traditional system considered in this paper. In fact, its objective is equivalent to minimizing the weighted number of late jobs, where the weight of job J_i in this case is given by $w_i v_i$. The later problem has been shown to be NP-hard [12], and Lawler [19] proposed a pseudo polynomial algorithm based on dynamic programming. The online version of this problem has also been studied extensively [5], [15]–[18], [25]. In contrast to Tians scheduling, this model does not consider partial results.

Partial evaluation: Application-level adaptation in web services offers interesting insights on how to produce flexible software that supports partial results [7], [11]. This line of work focuses on adapting the service quality based on the client capabilities. For example, a server may send images of different quality to clients based on the available bandwidth at each client. Different from our work, that work does not take advantage of the quality profile of best-effort applications to improve server capacity and they didn't consider scheduling.

Related to web search engines, Baek and Chilimbi [1] proposed a framework with multiple implementations of the search engine with various degrees of approximations based on partial executions. They studied the tradeoff between quality and performance and between quality and energy consumption, while we focus on Tians scheduling which should improve both metrics. They did not study scheduling.

Partial results with linear quality functions have been studied by [6], [9], [10]. In particular, executing a job with weight v_i for p_i amount of time, where $p_i \leq w_i$, gains quality $p_i v_i$. Hence, quality is linear function of job processing time, and each job has a weight. Chang and Yap [6] first considered this problem in the context of thinwire visualization and introduced two online algorithms, which were later improved [8]–[10]. Chrobak et al. [10] also proposed an offline optimal algorithm for this problem based on bipartite matching and maximum flow with time complexity $O(n^5)$. While these models consider quality profiles that are linear functions for weighted jobs, Tians targets different environment, namely large scale interactive services, in which jobs have equal weight concave quality profiles.

Other domains: Tians also shares some properties with related work from a few other domains. For example, partial execution has been used in anytime algorithms in AI [27]. Unlike most algorithms that run to completion, anytime algorithms provide a single answer after performing some fixed amount of computation. They are proposed to give intelligent systems the ability to return results of better quality in return longer response times.

A couple of other scheduling problems have also influenced Tians' design. Firstly, some previous work [2], [20], [26] considered scheduling jobs with deadlines on a processor with dynamic speed scaling capability. Their model assumes that the power consumption is a convex function of the processor speed, and the objective is to minimize the overall energy consumption. This energy minimization problem on convex function shares some similarity with our quality maximization problem on concave quality profile. Although partial execution is not considered, an offline algorithm [26] inspired us to apply divide-and-conquer approach in Tians-Optimal. Secondly, Tians uses SOEP to schedule jobs in a segment by allocating to each job an equal amount of time unless the job demands less. This shares similarities to dynamic equi-partitioning (DEQ) [22], which is used to schedule parallel jobs on multiprocessors where a job may get fewer processors than its parallelism with the tradeoff of increased execution time. DEQ considers neither deadline nor partial execution.

Related work on VOD systems: Some prior work [3], [4], [13] focused on providing admission control to decide whether or not to serve a streaming request to completion upon the request's arrival. While the objective is also to maximize the overall service quality, fine-grained dynamic decisions are not considered in trading off streaming quality with the available resources, as in Tians scheduling.

VIII. Conclusion Remarks

Large scale interactive services require good and predictable response quality. This is usually achieved by keeping servers lightly loaded. In this paper, we point out that in best-effort services, the response of a request has a quality profile. We introduce the Tians scheduling model to formalize this observation. In Tians scheduling, server software can produce partial results, and a scheduler assigns processing times to maximize response quality for interactive applications. We propose three scheduling algorithms: offline, online clairvoyant, and online nonclairvoyant. We prove the optimality of the offline scheduling algorithm and evaluate the two online scheduling algorithms using a simulation study. We simulate allocating CPU processing times in a

web search engine and allocating upstream bandwidth in a video-on-demand environment. Simulation results show a significant improvement of Tians scheduling over traditional server models without partial results: average response quality improves and the variance of responses decreases.

Acknowledgement

We thank Jim Larus, Burton Smith, Trishul Chilimbi, Albert Greenberg for helpful discussions and feedback.

References

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

[2] N. Bansal, T. Kimbrel, and K. Pruhs. Dynamic speed scaling to manage energy and temperature. In *FOCS*, 2004.

[3] A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour, and B. Schieber. Bandwidth allocation with preemption. In *STOC*, 1995.

[4] A. Bar-Noy, J. A. Garay, and A. Herzberg. Sharing video on demand. *Discrete Applied Mathematics*, 129:3–30, 2003.

[5] S. Baruah, G. Koren, D. Mao, B. Mishra, A. Raghunathan, L. Rosier, D. Shasha, and F. Wang. On the competitiveness of on-line real-time task scheduling. *Real-Time Systems*, 4:125–144, May 1992.

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

[7] Y. Chen. Detecting web page structure for adaptive viewing on small form factor devices. In *WWW*, pages 225–233, 2003.

[8] F. Y. L. Chin and S. P. Y. Fung. Improved competitive algorithms for online scheduling with partial job values. In *Computing and Combinatorics*, 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] A. Fox, S. D. Gribble, Y. Chawathe, and E. A. Brewer. Mobility. chapter Adapting to network and client variation using infrastructural process proxies: lessons and perspectives, pages 431–446. 1999.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.

[13] M. Garofalakis, Y. Ioannidis, B. Ozden, and A. Silberschatz. Competitive on-line scheduling of continuous-media streams. *Journal of Computer and System Sciences*, 64:219–248, 2002.

[14] C. Huang, P. A. Chou, and A. Klemets. Optimal control of multiple bit rates for streaming media. In *Picture Coding Symposium*, 2004.

[15] B. Kalyanasundaram and K. Pruhs. Speed is as powerful as clairvoyance. *Journal of the ACM*, 47(4):617–643, 2000.

[16] C.-Y. Koo, T. W. Lam, T.-W. Ngan, and K.-K. To. On-line scheduling with tight deadlines. In *Mathematical Foundations of Computer Science*, 2001.

[17] G. Koren and D. Shasha. Dover: An optimal on-line scheduling algorithm for overloaded uniprocessor real-time systems. *SIAM Journal on Computing*, 24:318–339, 1995.

[18] T. W. Lam and K. K. To. Trade-offs between speed and processor in hard-deadline scheduling. In *SODA*, 1999.

[19] E. L. Lawler. A dynamic programming algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Annals of Operations Research*, 26:125–133, 1991.

[20] M. Li, A. C. Yao, and F. F. Yao. Discrete and continuous min-energy schedules for variable voltage processors. *National Academy of Sciences*, 103:3983–3987, 2006.

[21] J. Liu, R. S.G., B. Li, and H. Zhang. Opportunities and challenges of peer-to-peer internet video broadcast. *Proceedings of the IEEE*, 96:11–24, 2008.

[22] C. McCann, R. Vaswani, and J. Zahorjan. A dynamic processor allocation policy for multiprogrammed shared-memory multiprocessors. *ACM Transactions on Computer Systems*, 11(2):146–178, 1993.

[23] R. Rejaie, M. Handley, and D. Estrin. Quality adaptation for congestion controlled video playback over the internet. *SIGCOMM Computer Communication Review*, 29:189–200, 1999.

[24] K. Sripanidkulchai, A. Ganjam, B. Maggs, and H. Zhang. The feasibility of supporting large-scale live streaming applications with dynamic application endpoints. *SIGCOMM Comput. Commun. Rev.*, 34:107–120, 2004.

[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] F. Yao, A. Demers, and S. Shenker. A scheduling model for reduced CPU energy. In *FOCS*, 1995.

[27] S. Zilberstein. Using anytime algorithms in intelligent systems. *AI Magazine*, 17(3):73–83, 1996.