Provably Delay Efficient Data Retrieving in Storage Clouds

Yin Sun[†], Zizhan Zheng[†], C. Emre Koksal[†], Kyu-Han Kim[‡], and Ness B. Shroff[†§]
[†]Dept. of ECE, [§]Dept. of CSE, The Ohio State University, Columbus, OH
[‡]Hewlett Packard Laboratories, Palo Alto, CA

***Abstract*—One key requirement for storage clouds is to be able to retrieve data quickly. Recent system measurements have shown that the data retrieving delay in storage clouds is highly variable, which may result in a long latency tail. One crucial idea to improve the delay performance is to retrieve multiple data copies by using parallel downloading threads. However, how to optimally schedule these downloading threads to minimize the data retrieving delay remains to be an important open problem. In this paper, we develop low-complexity thread scheduling policies for several important classes of data downloading time distributions, and prove that these policies are either delay-optimal or within a constant gap from the optimum delay performance. These theoretical results hold for an arbitrary arrival process of read requests that may contain finite or infinite read requests, and for heterogeneous MDS storage codes that can support diverse storage redundancy and reliability requirements for different data files. Our numerical results show that the delay performance of the proposed policies is significantly better than that of First-Come-First-Served (FCFS) policies considered in prior work.**

I. Introduction

Cloud storage is a prevalent solution for online data storage, as it provides the appealing benefits of easy access, low maintenance, elasticity, and scalability. The global cloud storage market is expected to reach $56.57 billion by 2019, with a compound annual growth rate of 33.1% [1].

In cloud storage systems, multiple copies of data are generated using simple replications [2]–[4] or erasure storage codes [5]–[8], and distributedly stored in disks, in-memory databases and caches. For an (n, k) erasure code $(n > k)$, data is divided into k equal-size chunks, which are then encoded into n chunks and stored in n distinct storage devices. If the code satisfies the typical maximum distance separable (MDS) property, any k out of the n chunks are sufficient to restore original data. When $k = 1$, the (n, k) erasure code reduces to the case of data replication (aka repetition codes).

Current storage clouds jointly utilize multiple erasure codes to support diverse storage redundancy and reliability requirements. For instance, in Facebook's data warehouse cluster, frequently accessed data (or so called "hot data") is stored with 3 replicas, while rarely accessed data ("cold data") is stored by using a more compressed (14,10) Reed-Solomon code to save space [6]. Open-source cloud storage softwares, such as HDFS-RAID [7] and OpenStack Swift [8], have been developed to support the coexistence of multiple erasure codes.

One key design principle of cloud storage systems is fast data retrieval. Amazon, Microsoft, and Google all report that a slight increase in user-perceived delay will result in a concrete revenue loss [9], [10]. However, in current storage clouds, data retrieving time is highly random and may have a long latency tail due to many reasons, including network congestion, load dynamics, cache misses, database blocking, disk I/O interference, update/maintenance activities, and unpredictable failures [2], [11]–[14]. One important approach to curb this randomness is *downloading multiple data copies in parallel*. For example, if a file is stored with an (n, k) erasure code, the system can schedule more than k downloading "threads", each representing a TCP connection, to retrieve the file. The first k successfully downloaded chunks are sufficient to restore the file, and the excess downloading threads are terminated to release the networking resources. By this, the retrieval latency of the file is reduced. However, scheduling redundant threads will increase the system load, which may in turn increase the latency. Such a policy provides a tradeoff between faster retrieval of each file and the extra system load for downloading redundant chunks. Therefore, a critical question is "how to optimally manage the downloading threads to minimize average data retrieving delay?" Standard tools in scheduling and queueing theories, e.g., [15]–[19] and the references therein, cannot be directly applied to resolve this challenge because they do not allow scheduling redundant and parallel resources for service acceleration.

In this paper, we rigorously analyze the fundamental delay limits of storage clouds. We develop low-complexity online thread scheduling policies for several important classes of data downloading time distributions, and prove that these policies are either delay-optimal or within a constant gap from the optimum delay performance.[1] Our theoretical results hold for an arbitrary arrival process of read requests that may contain finite or infinite read requests, and for heterogeneous MDS storage codes that can support diverse code parameters (n_i, k_i) for different data files. The main contributions of our paper are listed as follows and summarized in Table I. An interesting *state evolution* argument is developed in this work, which is essential for establishing the constant delay gaps; the interested reader is referred to the technical report [20] for the details.

- When the downloading times of data chunks are *i.i.d.* exponential with mean $1/\mu$, we propose a Shortest Expected Remaining Processing Time policy with Redundant thread assignment (SERPT-R), and prove that SERPT-

[1]By constant delay gap, we mean that the delay gap is bounded by a constant value that is independent of the request arrival process and system traffic load.

This work has been supported in part by an IRP grant from HP.

Theorem	Arrival process	Parameters of MDS codes	Service preemption	Downloading time distribution	Policy	Delay gap from optimum
1	any	$d_{\min} \geq L$	allowed	*i.i.d.* exponential	SERPT-R	delay-optimal
2	any	any	allowed	*i.i.d.* exponential	SERPT-R	$\frac{1}{\mu}\sum_{l=d_{\min}}^{L-1}\frac{1}{l}$
3	any	$d_{\min} \geq L$	not allowed	*i.i.d.* exponential	SEDPT-R	$1/\mu$
4	any	any	not allowed	*i.i.d.* exponential	SEDPT-R	$\frac{1}{\mu}\left(\sum_{l=d_{\min}}^{L-1}\frac{1}{l}+1\right)$
5	any	any	not allowed	*i.i.d.* New-Longer-than-Used	SEDPT-NR	$O(\ln L/\mu)$
6	any	any	allowed	*i.i.d.* New-Longer-than-Used	SEDPT-WCR	$O(\ln L/\mu)$
7	any	$k_i = 1$, $d_{\min} \geq L$	not allowed	*i.i.d.* New-Shorter-than-Used	SEDPT-R	delay-optimal

TABLE I: Summary of the delay performance of our proposed policies under different settings, where $d_{\min}$ is the minimum distance among all MDS storage codes defined in (2), $1/\mu$ is the average chunk downloading time of each thread, and L is the number of downloading threads. The classes of "New-Longer-than-Used" and "New-Shorter-than-Used" distributions are defined in Section V. Note that the delay gaps in this table are independent of the request arrival process and system traffic load.

R is *delay-optimal* among *all* online policies, if (i) the storage redundancy is sufficiently high and (ii) preemption is allowed. If condition (i) is not satisfied, we show that under SERPT-R, the extra delay caused by low storage redundancy is no more than the average downloading time of $(\ln L + 1)$ chunks, i.e., $(\ln L + 1)/\mu$, where L is the number of downloading threads. (This delay gap grows slowly with respect to L, and is independent of the request arrival process and system traffic load.) Further, if preemption is not allowed, we propose a Shortest Expected Differentiable Processing Time policy with Redundant thread assignment (SEDPT-R), which has a delay gap of no more than the average downloading time of one chunk, i.e., $1/\mu$, compared to the delay-optimal policy.
- When the downloading times of data chunks are *i.i.d.* New-Longer-than-Used (NLU) (defined in Section V), we design a Shortest Expected Differentiable Processing Time policy with Work-Conserving Redundant thread assignment (SEDPT-WCR) for the preemptive case and a Shortest Expected Differentiable Processing Time policy with No Redundant thread assignment (SEDPT-NR) for the non-preemptive case. We show that, comparing with the delay-optimal policy, the delay gaps of preemptive SEDPT-WCR and non-preemptive SEDPT-NR are both of the order $O(\ln L/\mu)$.
- When the downloading times of data chunks are *i.i.d.* New-Shorter-than-Used (NSU) (defined in Section V), we prove that SEDPT-R is delay-optimal among all online policies, under the conditions that data is stored with repetition codes, storage redundancy is sufficiently high, and preemption is not allowed.

We note that the proposed SEDPT-type policies are different from the traditional Shortest Remaining Processing Time first (SRPT) policy, and have not been proposed in prior work.

II. Related Work

The idea of reducing delay via multiple parallel data transmissions has been explored empirically in various contexts [21]–[25]. More recently, theoretical analysis has been conducted to study the delay performance of data retrieval in distributed storage systems. One line of studies [26]–[31] were centered on the data retrieval from a small number of storage nodes, where the delay performance is limited by the service capability of individual storage nodes. It was shown in [26] that erasure storage codes can reduce the queueing delay compared to simple data replications. In [27], [28], delay bounds were provided for First-Come-First-Served (FCFS) policies with different numbers of redundant threads. In [29], a delay upper bound was obtained for FCFS policies under Poisson arrivals and arbitrary downloading time distribution, which was further used to derive a sub-optimal solution for jointly minimizing latency and storage cost. In [30], the authors established delay bounds for the classes of FCFS, preemptive and non-preemptive priority scheduling policies, when the downloading time is *i.i.d.* exponential. In [31], the authors studied when redundant threads can reduce delay (and when not), and designed optimal redundant thread scheduling policies among the class of FCFS policies.

The second line of researches [32]–[34] focus on large-scale storage clouds with a large number of storage nodes, where the delay performance is constrained by the available networking resources of the system. In [32], [33], the authors measured the chunk downloading time over the Amazon cloud storage system and proposed to adapt code parameters and the number of redundant threads to reduce delay. In [34], it was shown that FCFS with redundant thread assignment is delay-optimal among all online policies, under the assumptions of a single storage code, high storage redundancy and exponential downloading time distribution. Following this line of research, in this paper, we consider the more general scenarios with heterogonous storage codes, general level of storage redundancy, and non-exponential downloading time distributions, where neither FCFS nor priority scheduling is close to delay-optimal.

III. System Model

We consider a cloud storage system that is composed of one frond-end proxy server and a large number of distributed storage devices, as illustrated in Fig. 1. The proxy server enqueues the user requests and establishes TCP connections to fetch data from the storage devices. In practice, the proxy server also performs tasks such as format conversion, data compression, authentication and encryption.[2]

[2]Our results can be also used for systems with multiple proxy servers, where each read request is routed to a proxy server based on geometrical location, or determined by a round robin or random load balancing algorithm. More complicated load balancing algorithms will be studied in our future work.

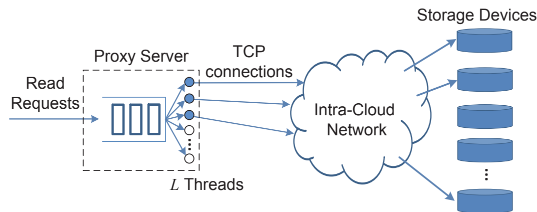


Fig. 1: System model.

A. Data Storage and Retrieval

Suppose that the file corresponding to request i is stored with an (n_i, k_i) MDS code.[3] Then, file i is partitioned into k_i equal-size chunks, which are encoded into n_i coded chunks and stored in n_i distinct devices. In MDS codes, any k_i out of the n_i coded chunks are sufficient to restore file i. Therefore, the cloud storage system can tolerate $n_i - k_i$ failures and still secure file i. Examples of MDS codes include repetition codes ($k_i = 1$) and Reed-Solomon codes. Let d_i denote the Hamming distance of an (n_i, k_i) MDS code, determined by

$$d_i = n_i - k_i + 1. \tag{1}$$

The minimum code distance of all storage codes is defined as

$$d_{\min} \triangleq \min\{d_i, i = 1, 2, \cdots\}. \tag{2}$$

It has been reported in [2], [11]–[14] that the downloading time of data chunks can be highly unpredictable in storage clouds. Some recent measurements [32]–[34] on Amazon AWS show that the downloading times of data chunks stored with distinct keys can be approximated as independent and identically distributed (*i.i.d.*) random variables. In this paper, we assume that the downloading times of data chunks are *i.i.d.*[4], as in [26]–[28], [31], [34].

B. Redundant and Parallel Thread Scheduling

The proxy server has L downloading threads, each representing a potential TCP connection, to retrieve data from the distributed storage devices. The value of L is chosen as the maximum number of simultaneous TCP connections that can occupy all the available networking bandwidth without significantly degrading the latency of each individual connection [32], [33]. A decision-maker at the proxy server determines which chunks to download and in what order for the L threads to minimize the average data retrieving delay.

Suppose that a sequence of N read requests arrive at the queue of the processing server.[5] Let a_i and $c_{i,\pi}$ denote the arrival and completion times of the ith request under policy π, respectively, where $0 = a_1 \leq a_2 \leq \cdots \leq a_N$. Thus, the service latency of request i is given by $c_{i,\pi} - a_i$, which includes both the downloading time and the waiting time in the request queue. We assume that the arrival process $(a_1, a_2, \cdots)$ is an *arbitrary deterministic* time sequence, while the departure process $(c_{1,\pi}, c_{2,\pi}, \cdots)$ is stochastic because of the random downloading time. Given the request parameters N and $(a_i, k_i, n_i)_{i=1}^N$, the average flow time of the requests under policy π is defined as

$$\overline{D}_\pi = \frac{1}{N} \sum_{i=1}^{N} \left(\mathbb{E}\left\{c_{i,\pi}\right\} - a_i \right), \tag{3}$$

where the expectation is taken with respect to the random distribution of chunk downloading time for given policy π and for given request parameters N and $(a_i, k_i, n_i)_{i=1}^N$. The goal of this paper is *to design low-complexity online thread scheduling policies that achieve optimal or near-optimal delay performance.*

Definition 1. Online policy: A scheduling policy is said to be *online* if, at any given time t, the decision-maker does not know the number of requests to arrive after time t, the parameters (a_i, k_i, n_i) of the requests to arrive, or the realizations of the (remaining) downloading times of the chunks that have not been accomplished by time t.

Definition 2. Delay-optimality: A thread scheduling policy π is said to be *delay-optimal* if, for *any* given request parameters N and $(a_i, k_i, n_i)_{i=1}^N$, it yields the shortest average flow time $\overline{D}_\pi$ among *all online* policies.

A key feature of this scheduling problem is the flexibility of *redundant and parallel thread scheduling*. Take file i as an example. When $n_i > k_i$, one can assign redundant threads to download more than k_i chunks of file i. The first k_i successfully downloaded chunks are sufficient for completing the read operation. After that, the extra downloading threads are terminated immediately, which is called *service termination*. By doing this, the retrieving delay of file i is reduced. On the other hand, redundant thread scheduling may cause extra system load. Therefore, such a policy provides a tradeoff between fast retrieving of each file and a potentially longer service latency due to the extra system load, which makes it difficult to achieve delay-optimality.

C. Service Preemption and Work Conserving

We consider *chunk-level* preemptive and non-preemptive policies. When preemption is allowed, a thread can switch to serve another chunk at any time, and resume to serve the previous chunk at a later time, continuing from the interrupted point. When preemption is not allowed, a thread must complete (or terminate) the current chunk before switching to serve another chunk. We assume that service terminations and preemptions are executed immediately with no extra delay.

Definition 3. Work-conserving: A scheduling policy is said to be *work-conserving* if all threads are kept busy whenever there are chunks waiting to be downloaded.

[3] The terms "file" and "request" are interchangeable in this paper.

[4] This assumption is reasonable for large-scale storage clouds, e.g., Amazon AWS, where individual read operations may experience long latency events, such as network congestion, cache misses, database blocking, high temperature or high I/O traffic of storage devices, that are unobservable and unpredictable by the decision-maker.

[5] The value of N can be either finite or infinite in this paper. If N tends to infinite, a $\limsup$ operator is enforced on the right hand side of (3).

Remark 1: *If preemption is allowed, a delay-optimal policy must be work-conserving*, because the average delay of any non-work-conserving policy can be reduced by assigning the idle threads to download more chunks. Meanwhile, *if preemption is not allowed, a work-conserving policy may not be delay-optimal*, because the occupied threads cannot be easily switched to serve an incoming request with a higher priority.

IV. Exponential Chunk Downloading Time

In this section, we study the delay-optimal thread scheduling when chunk downloading time is *i.i.d.* exponentially distributed with mean $1/\mu$. Non-exponential downloading time distributions will be investigated in Section V.

A. High Storage Redundancy, Preemption is Allowed

We first consider the case of high storage redundancy such that $d_{\min} \geq L$ is satisfied. In this case, we have $n_i - (k_i - 1) \geq L$ for all i. Hence, each file i has at least L available chunks even if $k_i - 1$ chunks of file i have been downloaded. Hence, each unfinished request has sufficient available chunks such that all L threads can be simultaneously assigned to serve this request.

Let s_j denote the arrival time of the jth arrived chunk downloading task of all files and t_j denote the completion time of the jth downloaded chunk of all files. The chunk arrival process $(s_1, s_2, \ldots)$ is uniquely determined by the request parameters $(a_i, k_i)_{i=1}^N$. Meanwhile, the chunk departure process $(t_1, t_2, \ldots)$ satisfies the following *invariant distribution* property:

Lemma 1. *[34, Theorem 6.4] Suppose that (i) $d_{\min} \geq L$ and (ii) the chunk downloading time is i.i.d. exponentially distributed with mean $1/\mu$. Then, for any given request parameters N and $(a_i, k_i, n_i)_{i=1}^N$, the distribution of the chunk departure process $(t_1, t_2, \ldots)$ is invariant under any work-conserving policy.*

We propose a preemptive Shortest Expected Remaining Processing Time first policy with Redundant thread assignment (**preemptive SERPT-R**):

Suppose that, at any time t, there are V unfinished requests $i_1, i_2, \ldots, i_V$, such that α_j chunks need to be downloaded for completing request i_j. Under SERPT-R, each idle thread is assigned to serve one available chunk of request i_j with the smallest α_j. (Due to storage redundancy, the number of available chunks of request i_j is larger than α_j.) If all the available chunks of request i_j are under service, then the idle thread is assigned to serve one available chunk of request $i_{j'}$ with the second smallest $\alpha_{j'}$. This procedure goes on, until all L threads are occupied or all the available chunks of the V unfinished requests are under services.

This policy is an extension of Shortest Remaining Processing Time first (SRPT) policy [15], [16] because it schedules parallel and redundant downloading threads to serve the requests with the least workload. The following theorem shows that this policy is delay-optimal under certain conditions.

Theorem 1. *Suppose that (i) $d_{\min} \geq L$, (ii) preemption is allowed, and (iii) the chunk downloading time is i.i.d. exponentially distributed with mean $1/\mu$. Then, for any given request parameters N and $(a_i, k_i, n_i)_{i=1}^N$, preemptive SERPT-R is delay-optimal among all online policies.*

Remark 2: Theorem 1 and the subsequent theoretical results of this paper are difficult to establish for the following reasons: 1) Each request i is partitioned into a batch of k_i chunk downloading tasks, and the processing time of each task is random. 2) There are $n_i - k_i$ redundant chunks for request i, such that completing any k_i of the n_i tasks would complete the request. 3) The system has L threads which can simultaneously process L tasks belonging to one or multiple requests. 4) If redundant downloading threads are scheduled, the associated extra system load must be considered when evaluating the delay performance.

Proof. We provide a proof sketch of Theorem 1. Consider an arbitrarily given chunk departure sample path $(t_1, t_2, \ldots)$. According to the property of the SRPT principle [15], [16], preemptive SERPT-R minimizes $\frac{1}{N}\sum_{i=1}^N (c_{i,\pi} - a_i)$ for any given sample path $(t_1, t_2, \ldots)$. Further, Lemma 1 tells us that the distribution of $(t_1, t_2, \ldots)$ is invariant among the class of work-conserving policies. By this, preemptive SERPT-R is delay-optimal among the class of work-conserving policies. Finally, since a delay-optimal policy must be work-conserving when preemption is allowed, Theorem 1 follows. See [20] for the details. □

In Theorem 6.4 of [34], it was shown that a First-Come-First-Served policy with Redundant thread assignment (FCFS-R) is delay-optimal when $k_i = k$ for all i and $d_{\min} \geq L$. In this case, preemptive SERPT-R reduces to the following policy: After a request departs from the system, pick any waiting request (not necessarily the request arrived the earliest) and assign all L threads to serve the available chunks of this request until it departs. Hence, FCFS-R belongs to the class of SERPT-R policies, and Theorem 6.4 of [34] is a special case of Theorem 1.

B. General Storage Redundancy, Preemption is Allowed

When $d_{\min} < L$, some requests may have less than L available chunks, such that not all of the L threads can be assigned to serve it. In this case, SERPT-R may not be delay-optimal. This is illustrated in the following example.

Example 1. *Consider two requests with parameters given as $(k_1 = 1, n_1 = 4, d_1 = 4, a_1 = 0)$ and $(k_2 = 2, n_2 = 2, d_2 = 1, a_2 = 0)$. The number of threads is $L = 4$. Under SERPT-R, all 4 threads are assigned to serve request 1 after time zero. However, after request 1 is completed, the chunk downloading rate is reduced from 4μ to 2μ, because request 2 only has $n_2 = 2$ chunks. Furthermore, after one chunk of request 2 is downloaded, the chunk downloading rate is reduced from 2μ to μ. The average flow time of SERPT-R is $\overline{D}_{\text{SERPT-R}} = 1/\mu$ seconds.*

We consider another policy Q: after time zero, 2 threads are assigned to serve request 1 and 2 threads are assigned to serve request 2. After the first chunk is downloaded, if the downloaded chunk belongs to request 1, then request 1 departs

and 2 threads are assigned to serve request 2. If the downloaded chunk belongs to request 2, then 3 threads are assigned to serve request 1 and 1 thread is assigned to serve request 2. After the second chunk is downloaded, only one request is left and the threads are assigned to serve the available chunks of this request. The average flow time of policy Q is $\overline{D}_Q = 61/(64\mu)$ seconds. Hence, SERPT-R is not delay-optimal.

Next, we bound the delay penalty associated with removing the condition $d_{\min} \geq L$.

Theorem 2. *If (i) preemption is allowed and (ii) the chunk downloading time is i.i.d. exponentially distributed with mean $1/\mu$. Then, for any given request parameters N and $(a_i, k_i, n_i)_{i=1}^N$, the average flow time of preemptive SERPT-R satisfies*

$$\overline{D}_{opt} \leq \overline{D}_{prmp,SERPT\text{-}R} \leq \overline{D}_{opt} + \frac{1}{\mu}\sum_{l=d_{\min}}^{L-1} \frac{1}{l}, \tag{4}$$

where $d_{\min}$ is defined in (2).

Proof. Here is a proof sketch of Theorem 2. We first use a *state evolution* argument to show that, after removing the condition $d_{\min} \geq L$, SERPT-R needs to download $L - d_{\min}$ or fewer additional chunks after any time t, so as to accomplish the same number of requests that are completed by SERPT-R with the condition $d_{\min} \geq L$ during $(0, t]$. Further, according to the properties of exponential distribution, the average time for the system to download $L - d_{\min}$ extra chunks under the conditions of Theorem 2 is upper bounded by the last term of (4). This completes the proof. See [20] for the details. □

Note that if $d_{\min} \geq L$, the last term in (4) becomes zero which corresponds to the case of Theorem 1; if $d_{\min} < L$, the last term in (4) is upper bounded by $\frac{1}{\mu}\left[\ln(\frac{L-1}{d_{\min}}) + 1\right]$. Therefore, the delay penalty caused by low storage redundancy is of the order $O(\ln L/\mu)$, and is insensitive to increasing L. Further, this delay penalty remains constant for any request arrival process and system traffic load.

C. High Storage Redundancy, Preemption is Not Allowed

Under preemptive SERPT-R, each thread can switch to serve another request at any time. However, when preemption is not allowed, a thread must complete or terminate the current chunk downloading task before switching to serve another request. In this case, SERPT-R may not be delay-optimal, as illustrated in the following example.

Example 2. *Consider two requests with parameters given as $(k_1 = 2, n_1 = 3, d_1 = 2, a_1 = 0)$ and $(k_2 = 1, n_2 = 2, d_2 = 2, a_2 = \varepsilon)$, where $\varepsilon > 0$ can be arbitrarily close to zero. The number of threads is $L = 2$, the chunk downloading time is i.i.d. exponentially distributed with mean $1/\mu$. Under SERPT-R, the two threads are assigned to serve request 1 after time zero. After the first chunk is downloaded, one thread is assigned to serve request 2 and the other thread remains to serve request 1. After the second chunk is downloaded, one of the requests has departed, and the two threads are assigned to serve the remaining request. The average flow time of SERPT-R is $\overline{D}_{SERPT\text{-}R} = 5/(4\mu) - \varepsilon/2$ seconds.*

We consider another non-preemptive policy Q: the threads remain idle until time ε. After ε, the two threads are assigned to serve request 2. After the first chunk is downloaded, request 2 has departed. Then, the two threads are assigned to serve request 1, until it departs. The average flow time of policy Q is $\overline{D}_Q = 1/\mu + \varepsilon/2$ seconds. Since ε is arbitrarily small, SERPT-R is not delay-optimal when preemption is not allowed.

We propose a non-preemptive Shortest Expected Differential Processing Time first policy with Redundant thread assignment (**non-preemptive SEDPT-R**), where the service priority of a file is determined by the difference between the number of remaining chunks of the file and the number of threads that has been assigned to the file.

Suppose that, at any time t, there are V unfinished requests $i_1, i_2, \ldots, i_V$, such that α_j chunks need to be downloaded for completing request i_j at time t and δ_j threads have been assigned to serve request i_j. Under non-preemptive SEDPT-R, each idle thread is assigned to serve one available chunk of request i_j with the smallest $\alpha_j - \delta_j$. (Due to storage redundancy, the number of available chunks of request i_j is larger than α_j. Hence, it may happen that $\alpha_j - \delta_j < 0$ because of redundant chunk downloading.) If all the available chunks of request i_j are under service, then the idle thread is assigned to serve one available chunk of request $i_{j'}$ with the second smallest $\alpha_{j'} - \delta_{j'}$. This procedure goes on, until all L threads are occupied or all the available chunks of the V unfinished requests are under services.

The intuition behind non-preemptive SEDPT-R is that δ_j chunks of request i_j will be under service after time t for any non-preemptive policy, and thus should be excluded when determining the service priority of request i_j. This is different from the traditional SRPT-type policies [15]–[19], which do not exclude the chunks under service when determining the service priorities of the requests. The delay performance of this policy is characterized in the following theorem:

Theorem 3. *Suppose that (i) $d_{\min} \geq L$, (ii) preemption is* ***not*** *allowed, and (iii) the chunk downloading time is i.i.d. exponentially distributed with mean $1/\mu$. Then, for any given request parameters N and $(a_i, k_i, n_i)_{i=1}^N$, the average flow time of non-preemptive SEDPT-R satisfies*

$$\overline{D}_{opt} \leq \overline{D}_{non\text{-}prmp,SEDPT\text{-}R} \leq \overline{D}_{opt} + 1/\mu. \tag{5}$$

Proof. We provide a proof sketch of Theorem 3. Theorem 1 tells us that preemptive SERPT-R provides a lower bound of $\overline{D}_{\text{opt}}$. On the other hand, non-preemptive SEDPT-R provides an upper bound of $\overline{D}_{\text{opt}}$. Thus, we need to show that the delay gap between preemptive SERPT-R and non-preemptive SEDPT-R is at most $1/\mu$. Towards this goal, we use a *state evolution* argument to show that for any time t and any given sample path of chunk departures $(t_1, t_2, \ldots)$, non-preemptive SEDPT-R needs to download L or fewer additional chunks after time t, so as to accomplish the same number of requests that are

completed under preemptive SERPT-R during $(0, t]$. By the properties of exponential distribution, the average time for the L threads to download L chunks under non-preemptive SEDPT-R is $1/\mu$, and Theorem 3 follows. See [20] for the details. □

Theorem 3 tells us that the delay gap between non-preemptive SEDPT-R and the optimal policy is at most the average downloading time of one chunk by each thread, i.e., $1/\mu$. Intuitively speaking, this is because each thread only needs to wait for downloading one chunk, before switching to serve another request. However, the proof of Theorem 3 is non-trivial, because it must work for any possible sample path of the downloading procedure.

D. General Storage Redundancy, Preemption is Not Allowed

When preemption is not allowed and the condition $d_{\min} \geq L$ is removed, we have the following result.

Theorem 4. *Suppose that (i) preemption is **not** allowed, and (ii) the chunk downloading time is i.i.d. exponentially distributed with mean $1/\mu$. Then, for any given request parameters N and $(a_i, k_i, n_i)_{i=1}^N$, the average flow time of non-preemptive SEDPT-R satisfies*

$$\overline{D}_{opt} \leq \overline{D}_{\text{non-prmp,SEDPT-R}} \leq \overline{D}_{opt} + \frac{1}{\mu} + \frac{1}{\mu}\sum_{l=d_{\min}}^{L-1}\frac{1}{l}, \tag{6}$$

where $d_{\min}$ is defined in (2).

If $d_{\min} \geq L$, the last term in (6) becomes zero which corresponds to the case of Theorem 3.

V. Non-Exponential Chunk Downloading Time

In this section, we consider two classes of general downloading time distributions: New-Longer-than-Used (NLU) distributions and New-Shorter-than-Used (NSU) distributions, defined as follows.[6]

Definition 4. New-Longer-than-Used distributions: A distribution on $[0, \infty)$ is said to be New-Longer-than-Used (NLU), if for all $t, \tau \geq 0$ and $\mathbb{P}(X > \tau) > 0$, the distribution satisfies

$$\mathbb{P}(X > t) \geq \mathbb{P}(X > t + \tau | X > \tau). \tag{7}$$

New-Shorter-than-Used distributions: A distribution on $[0, \infty)$ is said to be New-Shorter-than-Used (NSU), if for all $t, \tau \geq 0$ and $\mathbb{P}(X > \tau) > 0$, the distribution satisfies

$$\mathbb{P}(X > t) \leq \mathbb{P}(X > t + \tau | X > \tau). \tag{8}$$

NLU (NSU) distributions are closely related to log-concave (log-convex) distributions. Many commonly used distributions are NLU or NSU distributions [35]. In practice, NLU distributions can be used to characterize the scenarios where the downloading time is a constant value followed by a short latency tail. For instance, recent studies [32], [33] suggest that the data downloading time of Amazon AWS can be approximated as a constant delay plus an exponentially distributed random variable, which is an NLU distribution. On the other hand, NSU distributions can be used to characterize occasional slow responses resulting from TCP retransmissions, I/O interference, database blocking and/or even disk failures.

We will require the following definitions: Let $\vec{x} = (x_1, x_2, \ldots, x_m)$ and $\vec{y} = (y_1, y_2, \ldots, y_m)$ be two vectors in $\mathbb{R}^m$, then we denote $\vec{x} \leq \vec{y}$ if $x_i \leq y_i$ for $i = 1, 2, \ldots, m$.

Definition 5. Stochastic Ordering: [36] Let X and Y be two random variables. Then, X is said to be stochastically smaller than Y (denoted as $X \leq_{\text{st}} Y$), if

$$\mathbb{P}(X > t) \leq \mathbb{P}(Y > t) \text{ for all } t \in \mathbb{R}. \tag{9}$$

Definition 6. Multivariate Stochastic Ordering: [36] A set $U \subseteq \mathbb{R}^m$ is called *upper* if $\vec{y} \in U$ whenever $\vec{y} \geq \vec{x}$ and $\vec{x} \in U$. Let $\vec{X}$ and $\vec{Y}$ be two random vectors. Then, $\vec{X}$ is said to be stochastically smaller than $\vec{Y}$ (denoted as $\vec{X} \leq_{\text{st}} \vec{Y}$), if

$$\mathbb{P}(\vec{X} \in U) \leq \mathbb{P}(\vec{Y} \in U) \text{ for all upper sets } U \subseteq \mathbb{R}^m. \tag{10}$$

Stochastic ordering of stochastic processes (or infinite vectors) can be defined similarly [36].

A. NLU Chunk Downloading Time Distributions

We consider a non-preemptive Shortest Expected Differential Processing Time first policy with No Redundant thread assignment (**non-preemptive SEDPT-NR**):

Suppose that, at any time t, there are V unfinished requests $i_1, i_2, \ldots, i_V$, such that α_j chunks need to be downloaded for completing request i_j at time t and δ_j threads have been assigned to serve request i_j. Under non-preemptive SEDPT-NR, each idle thread is assigned to serve one available chunk of request i_j with the smallest $\alpha_j - \delta_j$. If α_j threads have been assigned to request i_j, then the idle thread is assigned to serve one available chunk of request $i_{j'}$ with the second smallest $\alpha_{j'} - \delta_{j'}$. This procedure goes on, until all L threads are occupied or each request i_j is served by α_j threads.

Note that since at most α_j threads are assigned to request i_j, we have $\alpha_j - \delta_j \geq 0$ for all i_j under non-preemptive SEDPT-NR. SEDPT-NR is a non-work-conserving policy. When preemption is allowed, the delay performance of SEDPT-NR can be improved by exploiting the idle threads to download redundant chunks. This leads to a preemptive Shortest Expected Differential Processing Time first policy with Work-Conserving Redundant thread assignment (**preemptive SEDPT-WCR**):

Upon the decision of SEDPT-NR, if each request i_j is served by α_j threads and there are still some idle threads, then assign these threads to download some redundant chunks to avoid idleness. When a new request arrives, the threads downloading redundant chunks will be preempted to serve the new arrival request.

Let us consider the service time for a thread to complete downloading one chunk. If the thread has spent τ seconds

[6] Note that New-Longer-than-Used (New-Shorter-than-Used) is **equivalent** to the term New-Better-than-Used (New-Worse-than-Used) used in reliability theory [35], [36], where "better" means a longer lifetime. However, this may lead to confusion in the current paper, where "better" means a shorter delay. We choose to use New-Longer-than-Used (New-Shorter-than-Used) to avoid confusion. In a recent work [31], the New-Longer-than-Used (New-Shorter-than-Used) property was termed light-everywhere (heavy-everywhere).

on one chunk, the tail probability for completing the current chunk under service is $\mathbb{P}(X > t + \tau | X > \tau)$. On the other hand, the tail probability for switching to serve a new chunk is $\mathbb{P}(X > t)$. Since the chunk downloading time is *i.i.d.* NLU, it is stochastically better to keep downloading the same chunk than switching to serve a new chunk.

Lemma 2. *Suppose that (i) the system load is high such that all L threads are occupied at all time $t \geq 0$ and (ii) the chunk downloading time is i.i.d. NLU. Then, for any given request parameters N and $(a_i, k_i, n_i)_{i=1}^N$, the chunk departure instants $(t_1, t_2, \ldots)$ under non-preemptive SEDPT-NR are stochastically smaller than those under any other online policy.*

Lemma 3. *Suppose that (i) the system load is high such that all L threads are occupied at all time $t \geq 0$, (ii) preemption is* **not** *allowed, and (iii) the chunk downloading time is i.i.d. NLU. Then, for any given request parameters N and $(a_i, k_i, n_i)_{i=1}^N$, the average flow time of non-preemptive SEDPT-NR satisfies*

$$\overline{D}_{opt} \leq \overline{D}_{\textit{non-prmp,SEDPT-NR}} \leq \overline{D}_{opt} + \mathbb{E}\left\{\max_{l=1,\ldots,L} X_l\right\}, \quad (11)$$

where the X_l's are i.i.d. chunk downloading times.

If the average chunk downloading time is $\mathbb{E}\{X_l\} = 1/\mu$, then the last term in (11) is bounded by

$$\frac{1}{\mu} \leq \mathbb{E}\left\{\max_{l=1,\ldots,L} X_l\right\} \leq \frac{1}{\mu}\sum_{l=1}^{L}\frac{1}{l}, \quad (12)$$

where the lower bound is trivial, and the upper bound follows from the property of New-Longer-than-Used distributions in Proposition 2 of [37]. Therefore, the delay gap in Lemma 3 is no more than $(\ln L + 1)/\mu$. Next, we remove condition (i) in Lemma 3 and obtain the following result.

Theorem 5. *Suppose that (i) preemption is* **not** *allowed and (ii) the chunk downloading time is i.i.d. NLU. Then, for any given request parameters N and $(a_i, k_i, n_i)_{i=1}^N$, the average flow time of non-preemptive SEDPT-NR satisfies*

$$\begin{aligned}\overline{D}_{opt} \leq \overline{D}_{\textit{non-prmp,SEDPT-NR}} &\leq \overline{D}_{opt} \\ &+\mathbb{E}\left\{\max_{l=1,\ldots,L} X_l\right\} + \mathbb{E}\left\{\max_{l=1,\ldots,L-1} X_l\right\}, \quad (13)\end{aligned}$$

where the X_l's are i.i.d. chunk downloading times.

When preemption is allowed, preemptive SEDPT-WCR can achieve a shorter average delay than non-preemptive SEDPT-NR. In this case, we have the following result.

Theorem 6. *Suppose that (i) preemption is allowed and (ii) the chunk downloading time is i.i.d. NLU. Then, for any given request parameters N and $(a_i, k_i, n_i)_{i=1}^N$, the average flow time of preemptive SEDPT-WCR satisfies*

$$\begin{aligned}\overline{D}_{opt} \leq \overline{D}_{\textit{prmt,SEDPT-WCR}} &\leq \overline{D}_{opt} \\ &+\mathbb{E}\left\{\max_{l=1,\ldots,L} X_l\right\} + \mathbb{E}\left\{\max_{l=1,\ldots,L-1} X_l\right\}, \quad (14)\end{aligned}$$

where the X_l's are i.i.d. chunk downloading times.

Similar to Lemma 3, the delay gaps in Theorems 5 and 6 are also of the order $O(\ln L/\mu)$.

B. NSU Chunk Downloading Time Distributions

If the chunk downloading time is *i.i.d.* NSU, one can show that it is stochastically better to switch to a new chunk than sticking to downloading the same chunk. We consider the scenario that preemption is not allowed and obtain the following result.

Lemma 4. *Suppose that (i) $d_{\min} \geq L$, (ii) $k_i = 1$ for all i, (iii) preemption is not allowed, and (iv) the chunk downloading time is i.i.d. NSU. Then, for any given request parameters N and $(a_i, k_i = 1, n_i)_{i=1}^N$, the chunk departure instants $(t_1, t_2, \ldots, t_N)$ under non-preemptive SEDPT-R are stochastically smaller than those under any other online policy.*

Theorem 7. *Suppose that (i) $d_{\min} \geq L$, (ii) $k_i = 1$ for all i, (iii) preemption is not allowed, and (iv) the chunk downloading time is i.i.d. NSU. Then, for any given request parameters N and $(a_i, k_i = 1, n_i)_{i=1}^N$, non-preemptive SEDPT-R is delay-optimal among all online policies.*

A special case of Theorem 7 was obtained in Theorem 3 of [31], where delay-optimality was shown only for high system load such that all L threads are occupied at all time.

VI. Numerical Results

We present some numerical results to illustrate the delay performance of different scheduling policies and validate the theoretical results. All these results are averaged over 100 random samples for the downloading times of data chunks.

A. Exponential Chunk Downloading Time Distributions

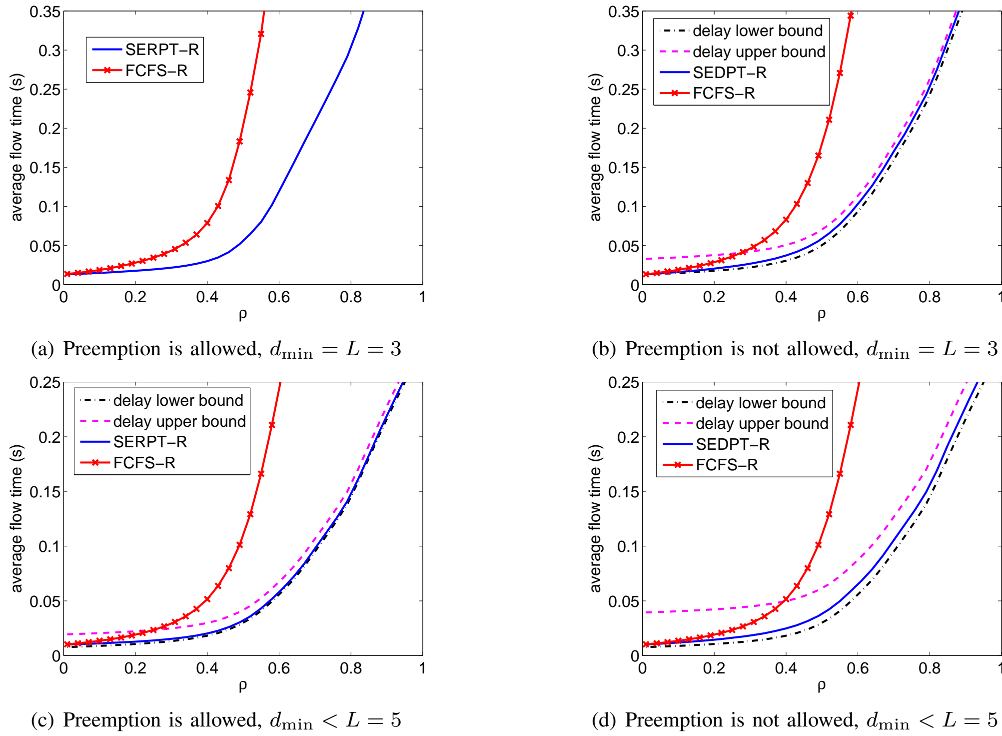
Consider a system with $N = 3000$ request arrivals, among which $p_1 = 90\%$ of the requested files are stored with a $(n_1, k_1, d_1) = (3, 1, 3)$ repetition code, and $p_2 = 10\%$ of the requested files are stored with a $(n_2, k_2, d_2) = (14, 10, 5)$ Reed-Solomon code. Therefore, $d_{\min} = 3$. The code parameters are drawn at random, *i.i.d.* from these two classes. The inter-arrival time of the requests is *i.i.d.* distributed as a mixture of exponentials:

$$X \sim \begin{cases} \text{Exponential(rate} = 0.5\lambda) & \text{with probability } 0.99; \\ \text{Exponential(rate} = 50.5\lambda) & \text{with probability } 0.01. \end{cases}$$

The average chunk downloading time is $1/\mu$ = 0.02s. The traffic intensity ρ is determined by

$$\rho = \frac{(p_1 k_1 + p_2 k_2)\,\lambda}{L\mu}. \quad (15)$$

Figures 2(a)-(d) illustrate the numerical results of average flow time $\overline{D}_\pi$ versus traffic intensity ρ for 4 scenarios where the chunk downloading time is *i.i.d.* exponentially distributed. One can observe that SERPT-R and SEDPT-R have shorter average flow times than the First-Come-First-Served policy with Redundant thread assignment (FCFS-R) [34]. If $L = d_{\min} = 3$ and preemption is allowed, by Theorem 1, preemptive SERPT-R is delay-optimal. For the other 3 scenarios, upper and lower



(a) Preemption is allowed, $d_{\min} = L = 3$

(b) Preemption is not allowed, $d_{\min} = L = 3$

(c) Preemption is allowed, $d_{\min} < L = 5$

(d) Preemption is not allowed, $d_{\min} < L = 5$

Fig. 2: Average flow time $\overline{D}_\pi$ versus traffic intensity ρ, where the chunk downloading time is *i.i.d.* exponentially distributed.

bounds of the optimum delay performance are plotted. By comparing with the delay lower bound, we find that the extra delay caused by non-preemption is 0.0114s which is smaller than $1/\mu = 0.02$s, and the extra delay caused by $d_{\min} < L$ is 0.0034s which is smaller than $\frac{1}{\mu}\sum_{l=d_{\min}}^{L-1}\frac{1}{l} = 0.0117$s. These results are in accordance with Theorems 1-4.

B. NLU Chunk Downloading Time Distributions

For the NLU distributions, the system setup is the same with that in the previous subsection. We assume that the chunk downloading time X is *i.i.d.* distributed as the sum of a constant and a value drawn from an exponential distribution:

$$\Pr(X > x) = \begin{cases} 1, & \text{if } x \leq \frac{0.4}{\mu}; \\ \exp\left[-\frac{\mu}{0.6}(x - \frac{0.4}{\mu})\right], & \text{if } x \geq \frac{0.4}{\mu}, \end{cases} \quad (16)$$

which was proposed in [32], [33] to model the data downloading time in Amazon AWS system. The traffic intensity ρ is also given by (15).

Figure 3 illustrates the average flow time $\overline{D}_\pi$ versus traffic intensity ρ when $L = 3$ and the chunk downloading time is *i.i.d.* NLU. As expected, preemptive SEDPT-WCR has a shorter average delay than non-preemptive SEDPT-NR. In the preemptive case, the delay performance of SEDPT-WCR is much better than those of non-preemptive SEDPT-R and the First-Come-First-Served policy with Work-Conserving Redundant thread assignment (FCFS-WCR). Therefore, preemptive SEDPT-WCR and non-preemptive SEDPT-NR are appropriate for *i.i.d.* NLU downloading time distributions. By comparing with the delay lower bound, we find that the maximum extra delays of preemptive SEDPT-WCR and non-preemptive SEDPT-NR are 0.0229s and 0.0230s, respectively. Both of them are smaller than the delay gap in Theorems 5 and 6, whose value is 0.0560s.

C. NSU Chunk Downloading Time Distributions

For NSU distributions, we consider that all $N = 3000$ requested files are stored with a $(n_1, k_1, d_1) = (3, 1, 3)$ repetition code. The chunk downloading time X is chosen *i.i.d.* as a mixture of exponentials:

$$X \sim \begin{cases} \text{Exponential}(\text{rate} = 0.4\mu) & \text{with probability } 0.5; \\ \text{Exponential}(\text{rate} = 1.6\mu) & \text{with probability } 0.5. \end{cases}$$

Under SEDPT-R, the average time for completing one chunk is $\mathbb{E}\{\min_{l=1,\cdots,L} X_l\}$, where the X_l's are *i.i.d.* chunk downloading times. Therefore, the traffic intensity ρ is

$$\rho = \lambda \mathbb{E}\left\{\min_{l=1,\cdots,L} X_l\right\}. \quad (17)$$

Figure 4 shows the average flow time $\overline{D}_\pi$ versus traffic intensity ρ where $L = 3$, preemption is not allowed, and the chunk downloading time is *i.i.d.* NSU. In this case, SEDPT-R is delay-optimal. We observe that the delay performance of SEDPT-WCR is quite bad and the delay gap between SEDPT-R and SEDPT-WCR is unbounded. This is because SEDPT-WCR has a smaller throughput region than SEDPT-R. Therefore, SEDPT-R is appropriate for *i.i.d.* NSU downloading time distributions.

VII. Conclusions

In this paper, we have analytically characterized the delay-optimality of data retrieving in distributed storage systems with multiple storage codes. Under several important settings, we

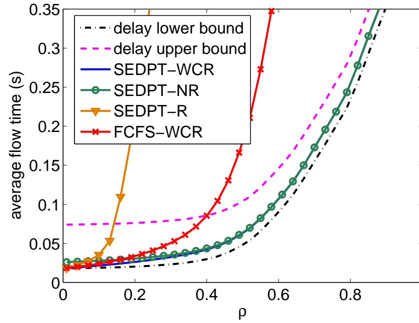


Fig. 3: Average flow time $\overline{D}_\pi$ versus traffic intensity ρ, where the chunk downloading time is *i.i.d.* NLU.

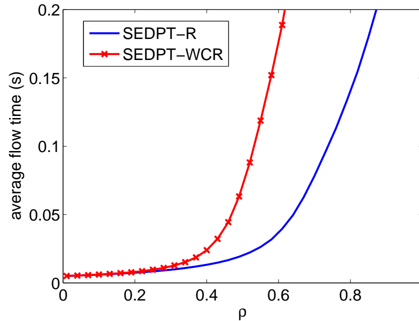


Fig. 4: Average flow time $\overline{D}_\pi$ versus traffic intensity ρ, where the chunk downloading time is *i.i.d.* NSU.

have shown that the proposed policies are either delay-optimal or within a constant gap from the optimum delay performance.

There are several important open problems concerning the analytical characterization of data retrieving delay:

- What is the optimal policy for other classes of non-exponential service distributions?
- What is the optimal policy when the service time distributions are heterogeneous across data chunks?
- What is the optimal policy when latency and downloading cost need to be jointly considered?
- How to design low-latency policies under delay metrics other than average flow time?

References

[1] "Public/private cloud storage market," http://www.marketsandmarkets.com/PressReleases/cloud-storage.asp.

[2] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *ACM SOSP*, 2003.

[3] F. Chang, J. Dean, S. Ghemawat, W. C. Hsieh, D. A. Wallach, M. Burrows, T. Chandra, A. Fikes, and R. E. Gruber, "Bigtable: A distributed storage system for structured data," in *Proc. of OSDI*, 2006.

[4] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The hadoop distributed file system," in *IEEE MSST*, 2010.

[5] A. Dimakis, K. Ramchandran, Y. Wu, and C. Suh, "A survey on network codes for distributed storage," *Proc. IEEE*, vol. 99, no. 3, pp. 476–489, 2011.

[6] K. V. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the facebook warehouse cluster," in *USENIX HotStorage*, 2013.

[7] HDFS-RAID wiki. [Online]. Available: http://wiki.apache.org/hadoop/HDFS-RAID.

[8] Swift-OpenStack wiki. [Online]. Available: https://wiki.openstack.org/wiki/Swift/.

[9] G. Linden, "Make data useful," Stanford CS345 Talk, 2006.

[10] E. Schurman and J. Brutlag, "The user and business impact of server delays, additional bytes, and HTTP chunking in web search," in *O'Reilly Velocity Web performance and operations conference*, 2009.

[11] J. Dean and L. A. Barroso, "The tail at scale," *Communications of the ACM*, vol. 56, no. 2, pp. 74–80, 2013.

[12] S. L. Garfinkel, "An evaluation of Amazon's grid computing services: EC2, S3 and SQS," Harvard University, Tech. Rep., 2007.

[13] G. Wang and T. S. E. Ng, "The impact of virtualization on network performance of amazon EC2 data center," in *IEEE INFOCOM*, 2010.

[14] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, and S. Yekhanin, "Erasure coding in windows azure storage," in *USENIX Annual Technical Conference*, Boston, MA, 2012, pp. 15–26.

[15] L. Schrage, "A proof of the optimality of the shortest remaining processing time discipline," *Operations Research*, vol. 16, pp. 687–690, 1968.

[16] D. R. Smith, "A new proof of the optimality of the shortest remaining processing time discipline," *Operations Research*, vol. 16, pp. 197–199, 1978.

[17] M. L. Pinedo, *Scheduling: Theory, Algorithms, and Systems*, 4th ed. Springer, 2012.

[18] K. Fox and B. Moseley, "Online scheduling on identical machines using srpt," in *SODA*, 2011.

[19] N. Bansal and K. Pruhs, "Server scheduling to balance priorities, fairness, and average quality of service," *SIAM Journal on Computing*, vol. 39, no. 7, pp. 3311–3335, 2010.

[20] Yin Sun, Zizhan Zheng, C. Emre Koksal, Kyu-Han Kim, and Ness B. Shroff, "Provably delay efficient data retrieving in storage clouds," http://arxiv.org/abs/1501.01661, 2014.

[21] S. Jain, M. Demmer, R. Patra, and K. Fall, "Using redundancy to cope with failures in a delay tolerant network," in *ACM SIGCOMM*, 2005.

[22] G. Ananthanarayanan, A. Ghodsi, S. Shenker, and I. Stoica, "Why let resources idle? aggressive cloning of jobs with dolly," in *USENIX HotCloud*, 2011.

[23] A. Vulimiri, O. Michel, P. B. Godfrey, and S. Shenker, "More is less: Reducing latency via redundancy," in *ACM HotNets*, 2012.

[24] A. Vulimiri, P. B. Godfrey, R. Mittal, J. Sherry, S. Ratnasamy, and S. Shenker, "Low latency via redundancy," in *ACM CoNEXT*, 2013.

[25] T. Flach, N. Dukkipati, A. Terzis, B. Raghavan, N. Cardwell, Y. Cheng, A. Jain, S. Hao, E. Katz-Bassett, and R. Govindan, "Reducing web latency: the virtue of gentle aggression," in *ACM SIGCOMM*, 2013.

[26] L. Huang, S. Pawar, H. Zhang, and K. Ramchandran, "Codes can reduce queueing delay in data centers," in *IEEE ISIT*, 2012.

[27] N. B. Shah, K. Lee, and K. Ramchandran, "The MDS queue: Analysing latency performance of codes," in *IEEE ISIT*, 2014.

[28] G. Joshi, Y. Liu, and E. Soljanin, "On the delay-storage trade-off in content download from coded distributed storage systems," *IEEE J. Sel. Areas Commun.*, vol. 32, pp. 989–997, May 2014.

[29] Y. Xiang, T. Lan, V. Aggarwal, and Y.-F. R. Chen, "Joint latency and cost optimization for erasure-coded data center storage," 2014. [Online]. Available: http://arxiv.org/abs/1404.4975

[30] A. Kumar, R. Tandon, and T. C. Clancy, "On the latency of erasure-coded cloud storage systems," 2014. [Online]. Available: http://arxiv.org/abs/1405.2833

[31] N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?" in *Allerton Conference*, 2013.

[32] G. Liang and U. Kozat, "FAST CLOUD: Pushing the envelope on delay performance of cloud storage with coding," *IEEE/ACM Trans. Networking*, 2013, in press.

[33] ——, "TOFEC: Achieving optimal throughput-delay trade-off of cloud storage using erasure codes," in *IEEE INFOCOM*, 2014.

[34] S. Chen, Y. Sun, U. Kozat, L. Huang, P. Sinha, G. Liang, X. Liu, and N. B. Shroff, "When queueing meets coding: Optimal-latency data retrieving scheme in storage clouds," in *IEEE INFOCOM*, 2014.

[35] M. Bagnoli and T. Bergstrom, "Log-concave probability and its applications," *Economic Theory*, vol. 26, no. 2, pp. 445–469, 2005.

[36] M. Shaked and J. G. Shanthikumar, *Stochastic Orders*. Springer, 2007.

[37] J. R. Berrendero and J. Cárcamo, "Characterizations of exponentiality within the HNBUE class and related tests," *Journal of Statistical Planning and Inference*, vol. 139, no. 7, pp. 2399–2406, 2009.