Novel Latency Bounds for Distributed Coded Storage

Parimal Parag*
*Department of Electrical Communication Engineering
Indian Institute of Science, Bengaluru, KA 560012, India
parimal@iisc.ac.in

Jean-Francois Chamberland†
†Department of Electrical and Computer Engineering
Texas A&M University, College Station, TX 77843-3128
chmbrlnd@tamu.edu

***Abstract*—Distributed storage systems are rapidly emerging as a desirable paradigm for cloud infrastructures. Through redundancy, such systems can improve the reliability of data backends. Interestingly, distributed storage can also enhance file access times for incoming requests, as it takes advantage of statistical averaging. Still, conducting performance characterization for ensuing delay profiles remains a challenge. Several recent contributions in this area construct bounds on the performance of redundant systems. These are then used to explore the latency-redundancy tradeoff, and assess the relative values of candidate implementations. Along these lines, this work establishes novel upper and lower bounds on the mean sojourn time of a request entering a distributed storage system. These bounds are based on stationary distributions of dominating quasi-birth-death processes and, in many settings, they can be made progressively tighter at the expense of additional computations.**

I. Introduction

Distributed storage systems are getting increasingly popular. This growing interest is rooted, partly, in the data deluge produced by large populations of inter-connected devices and the voracious appetite for media content of contemporary end-users. A significant portion of the data files downloaded on the Internet is hosted on content delivery networks and distributed storage infrastructures. These systems are utilized to ensure the wide and timely availability of content, protect against local failure through redundancy, and enable the statistical averaging of loads across time and geographical locations.

Forward error correcting codes are emerging as an integral part of distributed storage. Through the design of such codes, it is possible to control the size of file downloads, reconstruction traffic upon failure, and latency profiles for typical requests. For example, maximum distance separable (MDS) coding can improve dependability while limiting the amount of redundancy needed to attain a target level of performance [1]. Several recent and ongoing research initiatives are aimed at better understanding the limits and tradeoffs associated with distributed storage. These include efficient erasure correcting codes [2], [3], recovery schemes [4], repair traffic [5], [6], and security [7]–[9]. Key advances in the area have also created a need to understand the connections between redundancy and file download times [10]–[17]. Our article falls along these latter considerations, as we further explore the interplay between coding schemes and latency. In particular, the structure of a coding scheme dictates which collections of servers can be contacted for file recovery. These structures, combined with the random nature of processing times on Internet servers, yield distributions for request completion times. Various performance criteria can be derived from these ensuing distributions, such as the mean and variance of sojourn times, and the tail decay rate of a distribution.

At an elementary level, distributed coded storage relies on a few key notions. First, a media object is divided into k blocks of equal size and, subsequently, encoded into n pieces. Coded fragments are then stored at various geographical locations. When a file request arrives, the corresponding user must collect at least k data block from the caches to be able to reconstruct the content of the media object. Constraints on which k servers can be contacted depend on the coding scheme adopted by the storage infrastructure. Two emblematic strategies have emerged as benchmarks to capture the potential benefits of coding in the context of the latency-redundancy tradeoff. One approach, called *repetition coding*, is produced through simple file fragmentation and block replication. Under this rudimentary strategy, a request must gather one information piece of every possible kind to recover the original file. Once this is achieved, the media object is obtained by sorting and appending the various pieces. A more advanced paradigm is built on the fact that, over large fields, it is possible to create codewords that produce linearly independent subsets. For instance, under *MDS coding*, the original file can be recovered from any combination of k blocks via a standard decoding process. It may be pertinent to mention that, although we employ repetition and MDS coding as expository schemes, the focus is on developing tools for analysis rather than comparing the two schemes. With the rapid development of coding for distributed storage, performance analysis is of paramount importance to capture an evolving landscape which includes redundancy, locality, and latency.

The greater flexibility afforded by a more elaborate coding scheme can improve file recovery times. Yet, this process must be facilitated by a suitable download strategy. Coordination can be implemented in a centralized manner [17], or it can take

This material is based upon work supported by the National Science Foundation (NSF) under Grant No. CCF-1619085, and by the Science and Engineering Research Board (SERB) under Grant No. DSTO-1677. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF or the SERB.

place in a decentralized fashion where extraneous and obsolete jobs at caches are discarded whenever a user completes a file download [12], [18]. A subtlety associated with preemptive cancellation originates from the option to drop a job when exactly k block downloads are initiated or when k of them are finished. These two viewpoints are already present in the literature. In our work, we adopt the latter course of action; that is, superfluous jobs are terminated once a file request is complete and the user departs from the system.

In general, modeling a request queue for content stored over a distributed system can be quite challenging. Under typical Poisson arrivals and exponential service, the system admits a Markov chain representation with a countably infinite state space. Still, in such cases, the Markov models can remain difficult to analyze. As such, alternative solution paths are necessary. A candidate approach to understand such systems consists of finding tractable models that stochastically dominate the evolution of the actual Markov chains. For instance, the authors in [16], [17] devise tractable quasi-birth-death (QBD) processes that can bound the performance of the request queue using matrix-analytic methods [19]–[21]. Using this approach, one can efficiently evaluate key attributes of these dominating system and, hence, establish performance limits on the original queueing system. Likewise, the authors in [12], [13] introduce fork-join queues that bound the behavior of a request queue for redundant requests with cancellation. Implicitly, they study an MDS coding scheme where any size-k subset of servers can be contacted to execute a file download. The subordinate system is then utilized to derive bounds on the mean completion time.

A. Main Contributions

We revisit the distributed storage framework whereby file requests are processed by a collection of servers, each possessing one coded fragment. We consider the replication-cancellation scenario where redundant requests are only discarded after a service completion. We introduce a new analytical approach to study request completion delay. We then use this abstraction to establish novel upper and lower bounds on the latency performance of a distributed storage system with redundant requests. The performance of the dominating queues are amenable to efficient numerical evaluation, through matrix-geometric bounding techniques. More specifically, we expand the QBD methods introduced in [17] to scenarios where redundant requests are cancelled upon service completion. This necessitates the creation of new bounding processes, as a straightforward extension of existing results is not possible. Paralleling existing literature, the performance criteria we adopt are derived from the equilibrium distribution of the number of active requests in the system.

II. System Model

In this section, we introduce the mathematical model that we use to analyze the operation of distributed coded systems, along with relevant notation. As in established literature on the subject, we focus on the request queue associated with a single file m. This file is partitioned into k pieces, and we view these pieces as symbols in a large finite field $\mathbb{F}_q$,

$$m = (m_1, \ldots, m_k) \in \mathbb{F}_q^k.$$

Let C denote a linear code that maps a k-length media object into an n-length codeword,

$$C(m) = (C_1(m), \ldots, C_n(m)) \in \mathbb{F}_q^n.$$

The corresponding code rate is equal to k/n. For simplicity, we assume that its reciprocal, n/k, is an integer. As usual, the collection of all possible codewords forms the codebook, $\mathcal{C} = \{C(m) : m \in \mathbb{F}_q^k\}$.

The coded fragments of media object m are stored on the n servers. Specifically, we assume that $C_j(m)$ is stored at location j. At any point in time, a user may have downloaded a set of code blocks from the various servers. We refer to the indices of the downloaded blocks,

$$T \subseteq [n] \triangleq \{1, \ldots, n\},$$

as the *observed servers*. Keeping track of the servers which have contributed to T and of those eligible to provide useful fragments is key in understanding the evolution of the request queue.

A k-element subset $S \subseteq [n]$ is called an *information set* [22], if the original media object can be reconstructed unambiguously from $\mathcal{C}_S \triangleq \{(C_j(m) : j \in S) : C \in \mathcal{C}\}$, with no extraneous code fragments. The collection of all information sets for code C is given by

$$\mathcal{I}(C) \triangleq \{S \subseteq [n] : |S| = \dim(\operatorname{span}(\mathcal{C}_S)) = k\}.$$

For situations where the set of observed servers T is not an information set, we introduce the concept of *useful servers*,

$$\begin{aligned} M(T) &= \{j \notin T : j \in S \text{ for some } S \in \mathcal{I}(C)\} \\ &= \bigcup_{S \in \mathcal{I}(C)} S \setminus T. \end{aligned}$$

In words, $M(T)$ contains the indices of the servers which can deliver a useful piece of information to a request that has already gathered fragment set T.

Herein, we develop analysis techniques that apply to symmetric linear codes. That is, linear codes for which the cardinality of every set of useful servers depends solely on the number of independent fragments present in T. Based on this property, we introduce a convenient shorthand notation for the number of useful servers,

$$N_{|T|} \triangleq |M(T)|. \tag{1}$$

This condition may seem complicated, yet it is instrumental in establishing a desirable structure for the problem at hand. Moreover, the two emblematic coding paradigms discussed above, repetition and MDS coding, possess such a symmetry. At this point, it is appropriate to review these two classes of codes, C^{rep} and C^{mds} in the context of the terminology introduced thus far.

A. Repetition Coding

In some sense, repetition represents the simplest, non-trivial distributed coding scheme. Every piece m_κ is replicated and stored at n/k locations. A request can recover the original media object when it has collected one data fragment of each kind. When data block m_κ is stored at the servers with labels $\{s \in [n] : \lceil sk/n \rceil = \kappa\}$, the set of useful servers for C^{rep} can be written as

$$\mathcal{I}(C^{\text{rep}}) = \{S \subseteq [n] : |S| = k, \{s \in S : \lceil sk/n \rceil\} = [k]\}.$$

There are $(n/k)^k$ such information sets. Once a request has acquired fragment m_κ, no other server within $\{s \in S : \lceil sk/n \rceil = \kappa\}$ can provide it pertinent information. Thus, the set of useful servers for a request with fragments T is equal to

$$M^{\text{rep}}(T) = \bigcup_{j \notin T} \{s \in S : \lceil sk/n \rceil = j\}.$$

The number of useful servers is $N_{|T|} = (k - |T|)n/k$ for any information subset T.

Consider the scenario where $k = 3$ and $n = 6$. For repetition coding, there are three distinct components, which we label A, B, and C. Each of these fragments are stored at two locations. A request must successfully download one copy of each component before it departs. The operation of this implementation is illustrated in Fig. 1.

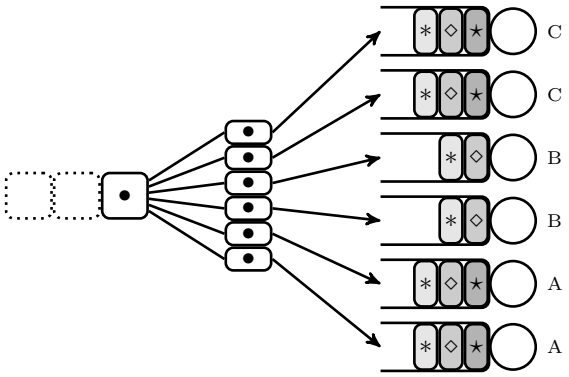

Figure 1. This diagram depicts a distributed fork-join network with six caches. Two servers are storing symbol A, two servers offer symbol B, and the remaining two servers host symbol C. Under this divide and replicate paradigm, a request must obtain piece A, piece B, and piece C to reconstruct the original media object.

B. MDS Coding

Contrastingly, MDS codes have optimal minimum Hamming distance, $d = n - k + 1$. Under this coding strategy, server $j \in [n]$ hosts fragment $C_j(m)$, which is unique. Moreover, every k distinct symbols form an information set, with

$$\mathcal{I}(C^{\text{mds}}) = \{S \subseteq [n] : |S| = k\}.$$

That is, a request can successfully recover the original media file by acquiring any k different data blocks. There are $|\mathcal{I}(C^{\text{mds}})| = \binom{n}{k}$ such sets. Thus, the MDS coding scheme features a larger number of information sets compared to repetition coding; naively, it should perform better. Suppose that a request has acquired segments T. Then, with MDS codes, every remaining server can offer a useful block;

$$M^{\text{mds}}(T) = [n] \setminus T.$$

where $T \subseteq S \in \mathcal{I}(C^{\text{mds}})$. The number of useful servers admits the simple form $N_{|T|} = n - |T|$.

Again, assume that $k = 3$ and $n = 6$. MDS coding produces unique fragments $\{C_j(m) : j \in [6]\}$, and each code block is stored at a different location. Moreover, a request can download pieces from any three servers to recover the original media object. Upon successful decoding, extraneous jobs are cancelled and the corresponding request departs from the system. This is shown in Fig. 2 for an MDS-based distributed storage system.

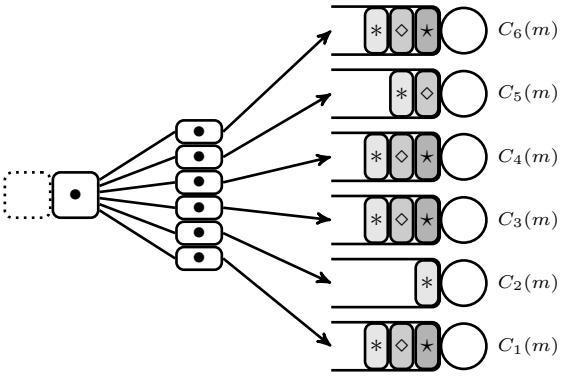

Figure 2. In a more elaborate fork-join system, the various caches store independent symbol of the coded message. The original media object can be recovered by decoding the content of any three distinct data blocks. Upon successful decoding, the corresponding user exits the system and its orphaned requests are dropped from the remaining queues.

III. Queueing Behavior

New requests for media object m arrive at a central location, and they are buffered in a queue of infinite capacity. All incoming users demand the entire file. We assume a Poisson arrival model. That is, the elapsed time between consecutive arrivals forms a sequence of independent and exponentially distributed random variables, each with parameter λ. Upon entering the system, a request seeks to aggregate k fragments from available servers as to form an information set. Once this is achieved, the request decodes these blocks, it recovers the original media object, and then departs from the queue. Extraneous downloads are cancelled immediately when the fragment they provide becomes irrelevant for the purpose of decoding.

Fragment downloads are facilitated by a group of homogenous servers. We assume that the service time at a particular location is an exponential random variable with rate μ, and it is independent over time and across caches. The waiting time between consecutive service opportunities also forms an instance of a Poisson process. Thus, the distributed storage infrastructure is a Poisson system. A direct consequence of

these modeling assumptions is the fact that, when a request is being served concurrently at multiple locations, the waiting time until its next download completion is the minimum of a collection of independent exponential random variables. As such, the effective waiting time is itself an exponential random variable with parameter $\varrho\mu$, where ϱ is the total number of caches where the request is being processed.

The overall operation of the queueing system we study matches the (n, k) *fork-join scheduling* policy defined in [12], [13]. To support the fork step every server has an individual queue, and each incoming request joins all n queues through replication. Data queries at a given location are attended to according to a first-come-first-served (FCFS) scheduling policy. In the join step, a request leaves the system as soon as it successfully gathers blocks associated with an information set and can therefore reconstruct the original file m. Pending block queries are dropped from useless queues as their parent request gathers new pieces. That is, as soon as a request acquires message subset T, all the servers in set $[n] \setminus M(T)$ abandon the block queries associated with the parent request. The task of matching requests to servers described above is stationary, work conserving, and it induces the Markov property. Coordination is established through a central agent. When a server completes a block query, it removes the request from its local buffer and notifies the central agent. As a notification arrives at the central agent, servers entering a useless set are instructed to discard their ongoing queries and to proceed with the next item. When a request gathers k pieces from an information set, all its remaining block queries are dropped and the request departs from the system.

A. Continuous-Time Markov Chain

The queueing system under consideration belongs to the class of Markov processes. One way to visualize its state space is to first consider the possibilities for T, the fragments gathered by a request. Let the collection of subsets of $[n]$ with cardinality less than k be denoted by

$$\mathcal{P}_{(n,k)} = \{S \subseteq [n] : |S| < k\}.$$

Then, for a work-conserving policy, we must have $T \in \mathcal{P}_{(n,k)}$. We represent the subset of blocks acquired by request i at time t by $S_i(t)$. Moreover, the number of requests in the system at time t is denoted by $r(t)$. When the system is empty of requests, we employ special symbol e to designate its state. Otherwise, we can express the state of the system at any time t as

$$\mathcal{S}(t) = (S_i(t) \in \mathcal{P}_{(n,k)} : i \in [r(t)]).$$

Admittedly, this state space is somewhat convoluted. Thus, we seek a simpler representation.

Theorem 1. *The collection of useful servers at time t,*

$$\{M(S_i(t)) : i \in [r(t)]\},$$

for a storage system with a symmetric code and fork-join FCFS scheduling is totally ordered by set inclusion. The sizes of the message subsets $\{|S_i(t)| : i \in r(t)\}$ form a monotone decreasing sequence in i.

Proof Sketch: Since the proof of Theorem 1 is lengthy and this result is not the main focus of our article, we only provide a brief outline for the demonstration; a more rigorous argument is omitted.

Stochastic process $\mathcal{S}(t)$ is a continuous-time Markov chain with a discrete state space. Consequently, it suffices to track transitions in its subordinate jump chain to establish the aforementioned properties. We present the argument below around one transition, and exclude the dependence on t for convenience. Recall that $M(S_i)$ is the set of useful servers for request i. Inductively, assume that symmetric coding combined with fork-join FCFS scheduling ensures that $M(S_{i-1}) \subseteq M(S_i)$ for all i. We check that this property is maintained through the next event. Suppose that server j intends to deliver a fragment to request i. Then, j must be contained in $M(S_i)$. Moreover, under FCFS, this is only possible when $j \notin M(S_{i-1})$. Thus, this transactions can only occur if

$$M(S_{i-1}) \subsetneq M(S_i)$$

and, necessarily, $|M(S_{i-1})| < |M(S_i)|$. This inequality, in turn, yields $|S_{i-1}| > |S_i|$. After the delivery of piece j to request i, we get

$$M(S_{i-1}) \subseteq M(S_i \cup \{j\})$$

and $|S_{i-1}| \geq |S_i \cup \{j\}|$. Since the statement of the theorem trivially holds at the onset of the process when the system is empty, and because the properties are maintained through arrivals and query completions, the theorem must be true at any time t. ∎

An important consequence of Theorem 1 is that it enables a much more intelligible representation for the state of the queueing system. Indeed, we gather that the number of useful servers and, hence, the rates at which requests are being served within the Markov chain depend exclusively on the cardinalities of the message subsets. This invites a system description based on state

$$L(t) = (L_i(t) = |S_i(t)| : i \in r(t)). \tag{2}$$

In words, it is enough to keep track of the number of data blocks accumulated by every request, rather than recording the labels of the blocks downloaded by each requests. Moreover, $L(t)$ inherits the Markov property from $\mathcal{S}(t)$ since transition rates in $L(t)$ are completely determined by its own state.

We take a closer look at possible states under $L(t)$. When $r(t) = 0$, then $L(t) = e$; this is rather uninteresting. On the other hand, suppose $r(t) = r > 0$. Then, we can employ vector notation $\ell = (\ell_1, \ldots, \ell_r)$ to denote admissible states. For a system with $r > 0$ active requests, these states are

$$\mathcal{L}_r = \{\{0, \ldots, k-1\}^r : \ell_i \geq \ell_{i+1}, i \in [r-1]\}. \tag{3}$$

The entire state space can then be summarized as

$$\mathcal{L} = \bigcup_{r \in \mathbb{N}_0} \mathcal{L}_r,$$

where $\mathcal{L}_0 = \{e\}$. With this deeper understanding of $L(t)$, we turn to transition rates.

B. Generator matrix

For any symmetric code, we can separate jumps into three categories. First, a new request may arrive; this event takes place at rate

$$\mathbf{Q}(\ell, (\ell, 0)) = \lambda.$$

We emphasize that λ is the parameter of the Poisson process governing arrivals in the system. Second, request i can obtain an additional data block from a server without leaving the system. Such an event is only possible when $r = \text{length}(\ell) \geq i$ and $\ell_i < k-1$. Under these circumstances, the corresponding transition rate is

$$\mathbf{Q}(\ell, \ell + e_i) = \left(N_{\ell_i} - N_{\ell_{i-1}}\right) \mu,$$

where N_{ℓ_i} is the number of useful servers to request i. For this equation to work with $i = 1$, we must use the convention $N_{\ell_0} = 0$; this convention is acceptable because there does not exist a request 0. The third possibility corresponds to the head request obtaining its last data block, which enables the decoding of the original media object and, consequently, leads to a departure from the queue. Such a transition can only happen if $\ell_1 = k - 1$ and, whenever this condition is met, it comes about with rate

$$\mathbf{Q}(\ell, (\ell_2, \ldots, \ell_r)) = N_{\ell_1} \mu.$$

Collectively, these rates determine the generator matrix for Markov chain $L(t)$. As usual, the diagonal elements of the generator matrix are defined such that the rows of the matrix sum to zero.

Having completely characterized $\mathbf{Q}$, we can theoretically simulate the evolution of $L(t)$. Moreover, since $\mathcal{S}(t)$ and $L(t)$ always have the same number of requests, we can assess the performance of a distributed storage system built around any symmetric code using $L(t)$. The caveat with this technique, of course, is the size and complexity of state space $\mathcal{L}$; they rapidly make analysis intractable. As mentioned in the introduction, our approach is to find computationally efficient means to bound the performance of coded systems. This way, we circumvent the challenges associated with getting stationary distributions for $L(t)$. This is accomplished in the next section.

IV. Bounding Techniques

The rationale behind bounding techniques is that the actual distributed coded system is hard to analyze. The candidate approach we adopt to assess performance consists in finding stochastic processes that dominate the evolution of the continuous-time Markov chain $L(t)$, yet remain tractable. This solution path is successfully employed in [12], [13]. Therein, the authors bound the operation of distributed systems where job cancellation takes place once k jobs are completed. An analogous approach can be found in [16], [17] for distributed storage systems where a file request is fulfilled by contacting exactly k servers. The goal of this section is to expand the techniques introduced in the latter set of citations, which yield families of increasingly tight bounds on performance, and apply them to the former scenario where the concurrent processing of requests from a same user is unconstrained. This research direction should produce novel upper and lower bounds on the performance of distributed coded systems that advance the state-of-the-art. Specifically, we wish to introduce tractable quasi-birth-death (QBD) processes that are amenable to analysis and confine the evolution of the underlying continuous-time Markov chain $L(t)$. The appeal of constrained QBD processes lies in their repetitive structures and the fact that they are well suited for numerical evaluation. This will become manifest shortly.

We create bounding QBD Markov chains by constraining the number of users who can simultaneously be processed by the servers. We let the parameter θ designate the maximum number of partially fulfilled users. Conceptually, we can partition the entries in $L(t)$ into two groups. The first group consists of *eligible requests*, which are allowed to be scheduled on servers and can therefore acquire data blocks. There can be at most θ such requests. The second group, which is composed of all users awaiting eligibility, is unbounded. However, none of these requests have accumulated partial information about the original media object.

To discuss the evolution of the upcoming QBD processes, it is useful to discuss their common state space. We emphasize that, if only the first θ requests are eligible for service, then all entries in ℓ with index $i > \theta$ must be zero. When the system is empty, we still denote the state by e. Furthermore, when the system features fewer than θ requests, the admissible states remain unchanged. However, when the number of requests exceed limit θ, vector ℓ must assume the form $(\ell_1, \ldots, \ell_\theta, 0, \ldots, 0)$ because the ineligible requests cannot have accumulated data blocks. Mathematically, this results in a slightly obscure state space given by

$$\mathcal{L} = \{e\} \cup \left(\bigcup_{r=1}^{\theta} \mathcal{L}_r \right) \cup \left(\mathcal{L}_\theta \times \underbrace{\left\{ \{0\}^{r-\theta} : r > \theta \right\}}_{\text{vectors of zeros}} \right) \tag{4}$$

where $\mathcal{L}_r$ is defined in (3). Still, the underlying concept is simple: constraining the number of partial requests reduces the complexity of the state space.

A more appropriate representation for this state space is to keep track of the entries that are lower than θ, and simply count the number of ineligible request beyond this threshold. To this end, we introduce a bijection $f(\cdot)$ of the following form. For cases where $r > \theta$, we define the mapping

$$\begin{aligned} f(\ell) &= f\left((\ell_1, \ldots, \ell_\theta) \oplus (\ell_{\theta+1}, \ldots, \ell_r)\right) \\ &= ((\ell_1, \ldots, \ell_\theta), r - \theta) = (\ell_{[1:\theta]}, r - \theta). \end{aligned}$$

When $r \leq \theta$, function $f(\cdot)$ is assigned value

$$f(\ell) = ((\ell_1, \ldots, \ell_r), 0) = (\ell, 0).$$

Finally, $f(e) = (e, 0)$. After this transformation, the state space for the resulting Markov chain $\tilde{L}(t)$ takes on the natural form

$$\mathcal{L}^{\mathrm{qbd}} \triangleq \left(\bigcup_{r=0}^{\theta} (\mathcal{L}_r, 0) \right) \cup \{(\mathcal{L}_\theta, q) : q \in \mathbb{N}\}.$$

This bijection yields a more compact notation and, more importantly, it reveals the level structure of our eventual QBD processes. We refer to the first component in tuple (ℓ, q) as the status of the eligible requests; and, to the second component as its level. Again, q indicates the number of extra requests awaiting eligibility.

We employ π to represent the stationary distribution of the continuous-time Markov chain over $\mathcal{L}^{\mathrm{qbd}}$, when it exists. As is customary, we can decompose π in terms of levels,

$$\pi = (\pi_0, \pi_1, \ldots, \pi_q, \ldots). \tag{5}$$

Excluding the first element, all the sub-vectors $\{\pi_q : q \in \mathbb{N}\}$ have the same size, $\mathrm{length}(\pi_q) = |\mathcal{L}_\theta|$. At level zero, we get $\mathrm{length}(\pi_0) = \sum_{r=0}^{\theta} |\mathcal{L}_r|$. With this notation, we are ready to establish the bounding stochastic processes.

A. QBD Reservation-θ

The extended QBD Reservation-θ model, as its name suggests, is closely related to the bound introduced in [17]. Consider a variation of the distributed coded system where only θ requests are eligible for service at any one time. Every other request, if present in the system, must wait until a suitable number of departures occur before being eligible for scheduling at a server. It can be helpful to think of these extra requests as awaiting eligibility in an auxiliary buffer. This abstract model is illustrated in Fig. 3.

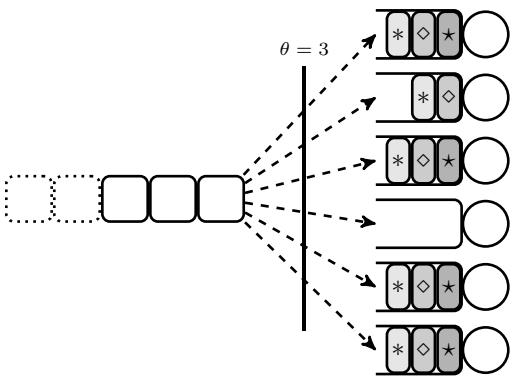

Figure 3. This block diagram showcases the operation of an extended QBD Reservation-3 system. In this case, only the first three requests are allowed to receive service. When the queues become imbalanced, a server may idle until the next departure, even though additional requests may be present in the system. Additional requests await eligibility in a generic buffer.

The difference in operation between QBD Reservation-θ and the standard system arises from instances of strong queue imbalance. Suppose there are θ partially fulfilled requests in the system and, at the same time, one of the servers is not useful to any of them. Under normal operation, this server initiates the processing of request $\theta+1$. Whereas, under QBD Reservation, this server idles until a departure occurs, as it is not allowed to fetch requests beyond threshold θ. This situation is shown in Fig. 3 for the threshold set at three concurrent requests. We denote the QBD Reservation-θ process by $\overline{L}(t)$.

The off-diagonal non-zero elements of generator matrix $\mathbf{Q}$ for $\overline{L}(t)$ can be inferred from the structure of the system. Transition rates corresponding to the arrivals are

$$\begin{aligned} \mathbf{Q}\left((\ell, 0), ((\ell, 0), 0)\right) &= \lambda \qquad r < \theta \\ \mathbf{Q}\left((\ell, q), (\ell, q+1)\right) &= \lambda \qquad r \geq \theta. \end{aligned}$$

Request completions and, necessarily, user departures can only occur when $\ell_1 = k - 1$; they are governed by

$$\begin{aligned} \mathbf{Q}\left((\ell, 0), ((\ell_2, \ldots, \ell_r), 0)\right) &= N_{\ell_1}\mu \qquad r \leq \theta \\ \mathbf{Q}\left((\ell, q), ((\ell_2, \ldots, \ell_\theta, 0), q-1)\right) &= N_{\ell_1}\mu \qquad r > \theta. \end{aligned}$$

Finally, all other non-vanishing rates have the form

$$\mathbf{Q}\left((\ell, q), (\ell + e_i, q)\right) = \left(N_{\ell_i} - N_{\ell_{i-1}}\right)\mu$$

where $\mathrm{length}(e_i) = \mathrm{length}(\ell)$ and $i \leq \min\{r, \theta\}$.

B. QBD Eviction-θ

We turn to the derivation of the lower bound, called QBD Eviction-θ. This lower bound is designed by adding extra help when a data fragment is destined to a request at level $(\theta+1)$. More precisely, suppose that the state of system $\underline{L}(t)$ is (ℓ, q) where both $q > 0$ and $\ell_\theta > 0$. Under normal system operation, $n - N_{\ell_\theta}$ servers would be processing jobs for request $(\theta+1)$. A query completion from one of these servers would then lead to $\ell_{\theta+1} = 1$. However, to preserve state space $\mathcal{L}^{\mathrm{qbd}}$ and lower bound system evolution, we simply assume that when the request $\ell_{\theta+1}$ receives this fragment, head request 1 is pushed out of the system at the same time. Equivalently, we can think of this situation as the head request instantly receiving enough information from a genie to decode the media object. Of course, this limits the number of partially fulfilled requests to θ, as desired. The operation of $\underline{L}(t)$ is depicted in Fig. 4.

We turn to the infinitesimal generator matrix $\mathbf{Q}$ associated with $\underline{L}(t)$. Its off-diagonal non-zero elements are identified below. As usual, transitions driven by an arrival take place at rates

$$\begin{aligned} \mathbf{Q}\left((\ell, 0), ((\ell, 0), 0)\right) &= \lambda \qquad r < \theta \\ \mathbf{Q}\left((\ell, q), (\ell, q+1)\right) &= \lambda \qquad r \geq \theta. \end{aligned}$$

Query completions directed at request i, where $i \in \{2, \ldots, \theta\}$ or where $i = 1$ with $\ell_1 < k - 1$, have rates

$$\mathbf{Q}\left((\ell, q), (\ell + e_i, q)\right) = \left(N_{\ell_i} - N_{\ell_{i-1}}\right)\mu;$$

above, $\mathrm{length}(e_i) = \mathrm{length}(\ell)$ and $i \leq \min\{r, \theta\}$. User departures are more difficult to describe for $\underline{L}(t)$. When $r \leq \theta$, a request completion can only happen when $\ell_1 = k - 1$; the corresponding rate in this case is

$$\mathbf{Q}\left((\ell, 0), ((\ell_2, \ldots, \ell_r), 0)\right) = N_{\ell_1}\mu.$$

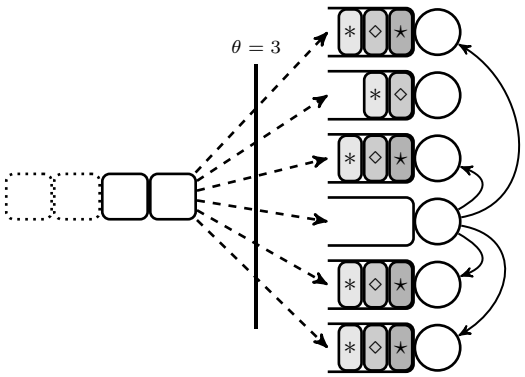

Figure 4. This diagram illustrates a QBD Eviction-3 process. For this system, only the first three requests are permitted to get service. If a server completes a query destined to request 4, a genie supplies enough information to push the head request out of the system, hence the name eviction. Thus, the number of partially fulfilled requests remains bounded by $\theta = 3$, while providing an optimistic lower bound on occupancy. As before, we can think of the extra requests as awaiting eligibility in a generic buffer.

Contrastingly, when $r \geq \theta$, a departure can arise naturally or it can take place under the auspice of the genie, with rates

$$\begin{aligned}\mathbf{Q}\left((\ell, q), ((\ell_2, \ldots, \ell_\theta, 0), q-1)\right) &= N_1\mu \\ \mathbf{Q}\left((\ell, q), ((\ell_2, \ldots, \ell_\theta, 1), q-1)\right) &= (n - N_\theta)\,\mu,\end{aligned}$$

respectively.

C. Block Structure

The book keeping involved in describing the infinitesimal generator matrices for $\overline{L}(t)$ and $\underline{L}(t)$ is admittedly tedious. The benefit of introducing these dominating processes, however, lies in the shared repetitive structure of $\mathbf{Q}$. For these two processes, the *level* refers to the number of requests awaiting scheduling eligibility. In an infinitesimal time duration, such a process can have, at most, a single jump. Hence, any transition resulting in a level change leads to a unit increase or decrease in the level. As such, we can define pertinent sub-matrices of sizes $|\mathcal{L}_\theta| \times |\mathcal{L}_\theta|$ that expose this structure. Suppose $q \geq 1$ and let $\ell, \check{\ell} \in \mathcal{L}_\theta$; we consider transitions originating from state (ℓ, q), with corresponding rates

$$\begin{aligned}\mathbf{A}_0(\ell, \check{\ell}) &\triangleq \mathbf{Q}((\ell, q), (\check{\ell}, q+1)) \\ \mathbf{A}_1(\ell, \check{\ell}) &\triangleq \mathbf{Q}((\ell, q), (\check{\ell}, q)) \\ \mathbf{A}_2(\ell, \check{\ell}) &\triangleq \mathbf{Q}((\ell, q), (\check{\ell}, q-1)).\end{aligned}$$

At the boundaries associated with level $q = 0$, more transitions are admissible. For these dynamics, and states $\ell, \check{\ell} \in \mathcal{L}_\theta$, $\ell_0, \check{\ell}_0 \in \cup_{r=0}^{\theta} \mathcal{L}_r$, we define

$$\begin{aligned}\mathbf{B}_{00}(\ell_0, \check{\ell}_0) &\triangleq \mathbf{Q}((\ell_0, 0), (\check{\ell}_0, 0)) \\ \mathbf{B}_{01}(\ell, \check{\ell}) &\triangleq \mathbf{Q}((\ell, 0), (\check{\ell}, 1)) \\ \mathbf{B}_{10}(\ell, \check{\ell}) &\triangleq \mathbf{Q}((\ell, 1), (\check{\ell}, 0)).\end{aligned}$$

Under proper state ordering, operator $\mathbf{Q}$ for either $\overline{L}(t)$ or $\underline{L}(t)$ has a semi-infinite matrix structure given by

$$\mathbf{Q} = \begin{bmatrix} \mathbf{B}_{00} & \mathbf{B}_{01} & \mathbf{0} & \mathbf{0} & \cdots \\ \mathbf{B}_{10} & \mathbf{A}_1 & \mathbf{A}_0 & \mathbf{0} & \cdots \\ \mathbf{0} & \mathbf{A}_2 & \mathbf{A}_1 & \mathbf{A}_0 & \cdots \\ \mathbf{0} & \mathbf{0} & \mathbf{A}_2 & \mathbf{A}_1 & \cdots \\ \vdots & \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{6}$$

where the submatrices $\mathbf{B}_{00}$, $\mathbf{B}_{01}$, $\mathbf{B}_{11}$, $\mathbf{A}_2$, $\mathbf{A}_1$, and $\mathbf{A}_0$ are determined by the level structure discussed above.

The continuous-time Markov processes associated with such $\mathbf{Q}$ belong to the class of random processes with repetitive structures. Thus, one can leverage numerical techniques from the vast literature on matrix-analytic methods and quasi-birth-death processes [19], [20]. The key insight behind this approach is to take advantage of the symmetric interactions among the different levels of the Markov chain. Specifically, for $q \geq 2$, the Chapman-Kolmogorov equations for the queueing system, $\pi\mathbf{Q} = \mathbf{0}$, yield

$$\pi_{q-1}\mathbf{A}_0 + \pi_q\mathbf{A}_1 + \pi_{q+1}\mathbf{A}_2 = \mathbf{0}$$

where π_q are the level components of π introduced in (5). In finding a solution to this matrix equation, the form of the embedded Markov structure and, specifically, its block partitioning are far more important than the values of each submatrix. The stationary distribution of the queue, when it exists, is characterized in the following theorem [19], [20], which is an adaptation of the matrix-geometric method to the current problem formulation.

Theorem 2. *Consider a positive recurrent and irreducible Markov chain on state space $\mathcal{L}^{\mathrm{qbd}}$ with transition probabilities given by* (6)*. Then, for stationary distribution π, there exists a constant matrix $\mathbf{R}$ such that*

$$\pi_q = \pi_{q-1}\mathbf{R} = \pi_1\mathbf{R}^{q-1}, \quad q \in 2, 3, \ldots.$$

Furthermore, constant matrix $\mathbf{R}$ fulfills

$$\mathbf{A}_0 + \mathbf{R}\mathbf{A}_1 + \mathbf{R}^2\mathbf{A}_2 = \mathbf{0}.$$

Matrix $\mathbf{R}$ may be computed as the limit, starting from $\mathbf{R}_0 = \mathbf{0}$, of the sequence defined by

$$\mathbf{R}_{p+1} = -\mathbf{A}_0\mathbf{A}_1^{-1} - \mathbf{R}_p\mathbf{A}_2\mathbf{A}_1^{-1}, \quad p = 1, 2, \ldots. \tag{7}$$

Let matrix $\tilde{\mathbf{Q}}$ be given by

$$\tilde{\mathbf{Q}} = \begin{bmatrix} \mathbf{B}_{00} & \mathbf{B}_{01} \\ \mathbf{B}_{10} & \mathbf{A}_1 + \mathbf{R}\mathbf{A}_2 \end{bmatrix}. \tag{8}$$

Then, $\tilde{\mathbf{Q}}$ is a rate matrix associated with an irreducible, finite Markov chain. If we denote the invariant distribution associated with $\tilde{\mathbf{Q}}$ by $(\tilde{\pi}_0, \tilde{\pi}_1)$, then the stationary distribution associated with $\mathbf{Q}$ can be expressed for $q \geq 1$ as

$$\begin{aligned}\pi_0 &= \frac{\tilde{\pi}_0}{(\tilde{\pi}_0 + \tilde{\pi}_1(\mathbf{I} - \mathbf{R})^{-1})\,\mathbf{1}}, \\ \pi_q &= \frac{\tilde{\pi}_1\mathbf{R}^{q-1}}{(\tilde{\pi}_0 + \tilde{\pi}_1(\mathbf{I} - \mathbf{R})^{-1})\,\mathbf{1}}.\end{aligned} \tag{9}$$

This theorem offers an algorithm to efficiently compute the stationary distributions associated with bounding processes $\overline{L}(t)$ and $\underline{L}(t)$. This is in stark contrast with the fact that there is no efficient way to compute the stationary distribution corresponding to $L(t)$. The bounds resulting from the application of the matrix-geometric methods, get progressively tighter as θ grows. Yet, from the structure of the theorem, we see how raising θ entails more computations.

V. Performance Evaluation

The characterization presented in the previous section makes it possible to efficiently compute a number of performance criteria. To enable a rapid comparison with previously published findings, we focus on the mean sojourn time of a request. We choose the random query time at a server to be distributed exponentially with rate $\mu = k/n$. With this convenient choice, we can present results as a function of normalized load. Data fragments for our distributed storage system are generated using a $(6, 3)$ linear code. We report findings for the two emblematic coding paradigms: repetition coding and MDS coding. We emphasize that, although the proposed QBD bounds can be computed for large system parameters, the selected values $k = 3$ and $n = 6$ enable the presentation of comparative simulation results.

We study the empirical average of sojourn time obtained by the simulation, and contrast it to the upper and lower bound on the mean value we obtain from the mean sojourn time in QBD Reservation-θ and Eviction-θ systems. In addition, we compare our proposed bounds to a closed form approximation and lower bound derived in [23], and a block service upper bound derived analytically in [12] for an MDS code. This upper bound has a connection to the extended QBD Reservation-1 bound, and can be generalized to other symmetric codes as well.

Figure 5 shows the two novel QBD bounds, along with simulation results, aforementioned approximation, and existing upper and lower bounds for repetition coding. The QBD bounds appear very good over all stable arrival rates. Not too surprisingly, the upper bound is much tighter than the block service upper bound. The QBD lower bound is very close to simulated performance at low loads, but it gets progressively looser at higher loads. This can be attributed to the greater propensity for the genie to evict requests when the queue is large. Also, the effect of an eviction is felt for the duration of a regenerative cycle for the request queue, i.e., until the queue length goes back to zero. Since higher loads imply longer generative cycles, the impact of the genie is more significant in this regime. A threshold $\theta = 12$ is suitable for exposition. For this rudimentary example, a much greater θ can be selected with reasonable computational complexity.

In Fig. 6, we plot the corresponding curves for MDS coding. In this case, the novel QBD bounds are very tight, and the threshold θ must be reduced to three to artificially create separation. For higher threshold values, the two bounds become nearly indistinguishable. This is very encouraging, as it points to the suitability of the proposed approach. The QBD

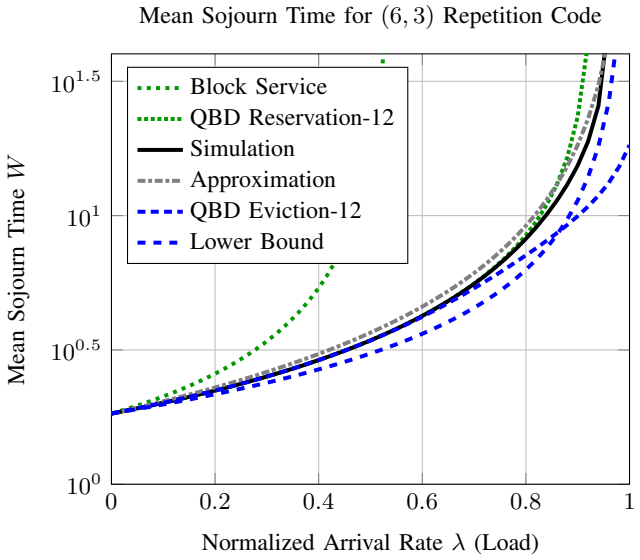

Figure 5. This graph showcases mean sojourn time for a $(6, 3)$ repetition scheme as a function of normalized arrival rate. The figure also contains the closed-form approximation, along with existing and novel bounds. The numerical QBD bounds can be made tighter by selecting a larger threshold θ; accuracy comes at the expense of additional computations.

bounds are much tighter for MDS coding; this stems from the fact that it is very difficult for the query queues associated with the servers to showcase strong imbalance. As such, the system rarely operates with a queue length larger than θ.

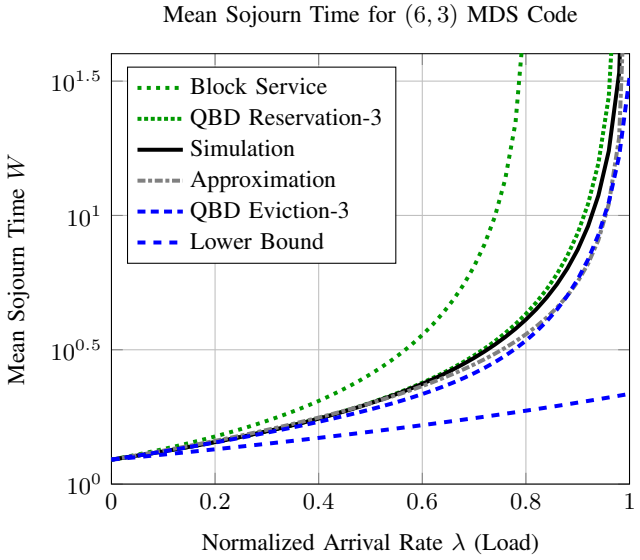

Figure 6. This chart exhibits mean sojourn time for a $(6, 3)$ MDS coding scheme as a function of normalized arrival rate. It also displays the closed-form approximation, along with the derived upper and lower bounds. The numerical QBD bounds are naturally tighter for MDS coding. Again in this case, they can be refined by increasing threshold θ.

Figure 7 compares the mean sojourn times of a request for repetition and MDS coding. As expected, the more elaborate MDS coded system outperforms repetition coding. The perfor-

mance gains are accurately predicted by the lower bounds on repetition coding and the upper bound on MDS coding, over a range of system loads. This is especially relevant for systems with intricate coding schemes and multiple servers because, in such situations, the actual system becomes hard to simulate. We included the Eviction-25 lower bound for repetition coding to illustrate how the bound gets tighter as θ increases.

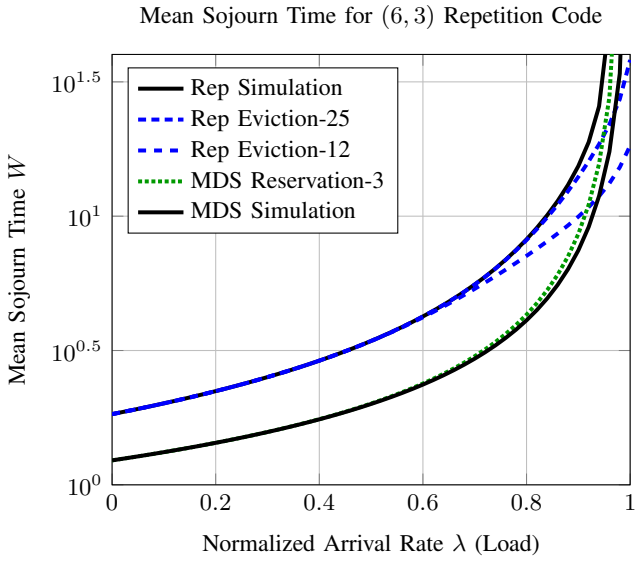

Figure 7. This plot provides a comparison of the MDS and repetition coding schemes. Due to its greater flexibility, the MDS coding scheme outperforms repetition coding, showcasing significant gains across loads. This trend is also accurately captured over most loads with the QBD bounding techniques proposed in this article.

VI. Concluding Remarks

In this work, we introduce a novel methodology to bound the performance of distributed coded systems based on dominating QBD processes. These bounding techniques apply to FCFS fork-join scheduling with symmetric codes and homogeneous servers. They advance the state-of-the-art, with significant improvements over existing bounds. And, they can be employed to characterize the latency-redundancy tradeoff for candidate implementations. We note that the QBD bounds can be made tighter by selecting a large parameter θ. Still, greater accuracy comes at the expense of additional computations.

Potential avenues for future research include relaxing the assumption regarding exponential service and independence across caches. Matrix-analytic methods have been applied to semi-Markov queueing models. Such extensions may find application in the context of distributed coded systems as well. In this article, we allude to the difference between cancelling requests when k jobs are scheduled or discarding them immediately after k jobs are completed. For non-exponential service, it would be interesting to delineate conditions under which one strategy outperforms the other. Finally, the field of distributed coded systems is evolving rapidly. It would be interesting to integrate recent developments such as locally repairable codes and secure codes into the latency-redundancy tradeoff.

References

[1] A. G. Dimakis, P. B. Godfrey, Y. Wu, M. J. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 4539–4551, 2010.

[2] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman, "Efficient erasure correcting codes," *IEEE Trans. Inf. Theory*, vol. 47, no. 2, pp. 569–584, 2001.

[3] A. Shokrollahi, "Raptor codes," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2551–2567, 2006.

[4] A. G. Dimakis, V. Prabhakaran, and K. Ramchandran, "Decentralized erasure codes for distributed networked storage," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2809–2816, June 2006.

[5] N. B. Shah, K. V. Rashmi, P. V. Kumar, and K. Ramchandran, "Explicit codes minimizing repair bandwidth for distributed storage," in *Information Theory Workshop (ITW)*. IEEE, 2010, pp. 1–5.

[6] S. El Rouayheb and K. Ramchandran, "Fractional repetition codes for repair in distributed storage systems," in *Allerton Conference on Communication, Control, and Computing*. IEEE, 2010, pp. 1510–1517.

[7] S. Jaggi, M. Langberg, S. Katti, T. Ho, D. Katabi, M. Médard, and M. Effros, "Resilient network coding in the presence of byzantine adversaries," *IEEE Trans. Inf. Theory*, vol. 54, no. 6, pp. 2596–2603, 2008.

[8] D. Wang, D. Silva, and F. R. Kschischang, "Robust network coding in the presence of untrusted nodes," *IEEE Trans. Inf. Theory*, vol. 56, no. 9, pp. 4532–4538, Sept 2010.

[9] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for distributed storage at the MSR and MBR points via a product-matrix construction," *IEEE Trans. Inf. Theory*, vol. 57, no. 8, pp. 5227–5239, Aug 2011.

[10] R. Rojas-Cessa, L. Cai, and T. Kijkanjanarat, "Scheduling memory access on a distributed cloud storage network," in *Wireless and Optical Communications Conference (WOCC)*, April 2012, pp. 71–76.

[11] S. Chen, Y. Sun, U. C. 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 Conference on Computer Communications*, April 2014, pp. 1042–1050.

[12] 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, no. 5, pp. 989–997, May 2014.

[13] G. Joshi, E. Soljanin, and G. W. Wornell, "Queues with redundancy: Latency-cost analysis," *SIGMETRICS Perform. Eval. Rev.*, vol. 43, no. 2, pp. 54–56, Sep. 2015.

[14] D. Wang, G. Joshi, and G. Wornell, "Efficient task replication for fast response times in parallel computation," in *International Conference on Measurement and Modeling of Computer Systems (SIGMETRICS)*. New York, NY, USA: ACM, 2014, pp. 599–600.

[15] B. Li, A. Ramamoorthy, and R. Srikant, "Mean-field-analysis of coding versus replication in cloud storage systems," in *IEEE International Conference on Computer Communications*. IEEE, 2016, pp. 1–9.

[16] N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?" *IEEE Trans. Commun.*, vol. 64, no. 2, pp. 715–722, 2016.

[17] K. Lee, N. B. Shah, L. Huang, and K. Ramchandran, "The MDS queue: Analysing the latency performance of erasure codes," *IEEE Trans. Inf. Theory*, vol. 63, no. 5, May 2017.

[18] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, E. Hyytiä, and A. Scheller-Wolf, "Queueing with redundant requests: exact analysis," *Queueing Systems*, vol. 83, no. 3, pp. 227–259, 2016.

[19] M. F. Neuts, *Matrix-Geometric Solutions in Stochastic Models: An Algorithmic Approach*. Dover Publications, 1995.

[20] G. Latouche and V. Ramaswami, *Introduction to Matrix Analytic Methods in Stochastic Modeling*, ser. ASA-SIAM Series on Statistics and Applied Probability. Society for Industrial Mathematics, 1987.

[21] R. V. Evans, "Geometric distribution in some two-dimensional queuing systems," *Operations Research*, vol. 15, no. 5, pp. 830–846, Sep.-Oct. 1967.

[22] P. Gopalan, G. Hu, S. Kopparty, S. Saraf, C. Wang, and S. Yekhanin, "Maximally recoverable codes for grid-like topologies," in *ACM-SIAM Symposium on Discrete Algorithms (SODA)*, Philadelphia, PA, USA, 2017, pp. 2092–2108.

[23] P. Parag, A. Bura, and J.-F. Chamberland, "Latency analysis for distributed storage," in *IEEE International Conference on Computer Communications*. IEEE, 2017, pp. 1–9.