Codes Can Reduce Queueing Delay in Data Centers

Longbo Huang, Sameer Pawar, Hao Zhang, Kannan Ramchandran

Abstract—In this paper, we quantify how much codes can reduce the data retrieval latency in storage systems. By combining a simple linear code with a novel request scheduling algorithm, which we call Blocking-one Scheduling (BoS), we show analytically that it is possible to reduce data retrieval delay by up to 17% over currently popular replication-based strategies. Although in this work we focus on a simplified setting where the storage system stores a single content, the methodology developed can be applied to more general settings with multiple contents. The results also offer insightful guidance to the design of storage systems in data centers and content distribution networks.

I. Introduction

In today's data centers, one of the most demanding tasks (in terms of latency) is "disk-read," e.g., fetching the data for performing analytics such as MapReduce, or to serve the data to the end consumer. In many cases, this task is greatly complicated by the highly non-uniform data popularity, where the most popular data objects can be accessed ten times more frequently than the bottom third [1]. This skewed demand leads to high contention for read tasks of the most popular data. To meet the demand and reduce data retrieval latency, current systems often introduces data redundancy by replicating many copies of each content, e.g., Hadoop replicates each content three or more times, to make the popular data more available and relieve the hot spots of read contentions, thereby reducing the average request latency.

This motivates us to *investigate the fundamental role of redundancy in improving the system latency*. In particular, we compare two systems, one using codes and the other using simple replication. Heuristically, codes offer more flexibility when retrieving the data from the servers, thus may improve content retrieval latency. In this paper, we use a *non-asymptotic analytical approach* to quantify this intuition.

To make the problem more concrete, consider an abstracted example of retrieving a file in a data center shown in Fig. 1. The storage system consists of 4 storage units, each capable of storing 1 packet of the desired file that consists of 2 packets A and B. We consider two possible storage strategies 1) each packet is replicated two times; 2) file is encoded using a $(4, 2)$ Maximum-Distance-Code (MDS) code. It is easy to see the redundancy factor is 2 in both cases. There is a common dispatcher that queues and schedules the incoming requests. We assume the request process is Poisson and the service time of the storage unit is exponentially distributed.

Quantifying the exact request delay in a coded system is a challenging task. The main difficulty is that *the scheduling algorithm needs to remember which SUs served earlier requests* to ensure that the request of a particular content

The authors are with the EECS dept at UC Berkeley, Berkeley, CA, 94720, USA. Emails: {huang, spawar, zhanghao, kannanr}@eecs.berkeley.edu.

is always served by distinct SUs. This makes the analysis extremely difficult. Moreover, since data centers cannot afford a redundancy factor of more than a few tens at most, the asymptotic analysis based approach advocated in [3] [9] are not relevent.

To tackle this challenge, we develop a novel scheduling algorithm called Blocking-one Scheduling (BoS). The idea is to block some subsequent content retrieval requests until the head-of-line request is processed. This helps remove the dependency in the scheduling actions, and allows a clean analysis of the delay performance of system using codes. Fortunately, as we will see later, the fraction of throughput loss due to BoS is $O(1/r^2)$, where r is the redundancy factor. Indeed, even for $r = 2$ in our simple example, the BoS achieves 96% of the maximum system throughput.

Under the above settings, we analytically show that strategy 2 that uses codes reduces the average request delay, for the considered example, by 7% or more compared to strategy 1 that uses replication. The intuition behind this is that in the replication system, the requests for packet A or B can be satisfied by only two specific servers, while in the coded system *any* two servers are sufficient to serve the file. Therefore, codes offer more flexibility in data retrieval due to multiplexing gain of available servers.
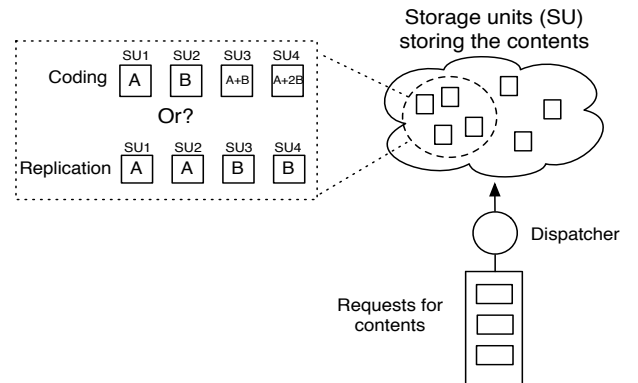

Fig. 1. An example information storage system storing a file that consists of two packets A and B. Requests for contents are assigned by a central dispatcher to available storage units. In order to reduce the content retrieval latency, we can either use *replication*, i.e., store the packets A, B twice, or *coding*, i.e., store coded packets of A and B.

Related Work: The authors in [5] and [11] study the throughput-delay gains of network coding in a single hop wireless downlink with unreliable channels. The authors of [8] consider the network coding gains in throughput when packets have hard delay deadlines. The work of [4] studies the gains in delay-throughput when using network coding over a linear network with unreliable links. In [10], the authors use network coding to prevent underflow in a multi-media streaming application. While most of the earlier works focus on use of codes to achieve the best effort throughput and

or delay over unreliable wireless channels, non-asymptotic and theoretical analysis of queueing delay in coded systems remains largely unaddressed, to the best of our knowledge.

The paper is organized as follows. In Section II, we state our system model. In Section III, we present the schemes that will be used in the uncoded and coded systems, and present the Blocking-one Scheduling (BoS) algorithm. We analyze the performance of BoS in Section IV. We then conclude the paper in Section V.

II. SYSTEM MODEL

We consider an information storage system that consists of n homogeneous storage servers, called storage units. Each SU has a storage capacity of one, and is capable of serving each incoming content request in a time that is exponentially distributed with mean $\mu = 1$. The system stores and serves a set of contents, denoted by $\mathcal{C} = \{1, 2, ..., C\}$. Each content is striped into k packets of size one. A total of k SU is needed to store a single content. In practice, for reliability and availability purposes, a content is stored redundantly on multiple servers.

Assume the *content retrieval requests* for each content c form a Poisson arrival process with rate λ_c. Since a content is striped into k packets that are stored on distinct SUs, every request needs to be served at k servers with distinct packets in order to fully retrieve the desired content. We model this behavior by duplicating each arrived request into k *packet-requests* and by making sure that no two packet-requests are processed by servers with identical packets.

There is a central dispatcher that delegates the incoming requests to the SUs. Upon arrival, the requests are first queued at the central dispatcher, and then sent to the servers once they become available[1]. In such a scenario, we are interested in quantifying the reduction in average content delay between a coding-based system and a replication-based system.

To illustrate the problem further, consider a simple example depicted in the Fig. 2. The system contains 4 storage units and 1 content. The content is striped into 2 packets A and B, and is stored with a redundancy factor of 2. Each arriving content request brings into the system two packet-requests, e.g., $3A, 3B$. In Fig. 2(a) we show a system that uses replication to introduce redundancy, where packets A and B are replicated twice. In contrast, Fig. 2(b) shows a system that uses a MDS code of rate 0.5 to introduce the desired redundancy. The question we aim to answer in this paper is *by how much can we reduce the average delay of a coding-based system over a replication-based one?*

In general, quantifying the delay performance of a system with multiple contents, multiple servers and multiple coded packets that are replicated multiple times can be quite challenging. The main difficulty lies in the fact that the SUs now store coded data, thus the scheduling algorithm has to ensure that the requests for the same content are not served by a same SU. This means that the scheduling algorithm may need to remember at which SUs all the earlier requests are
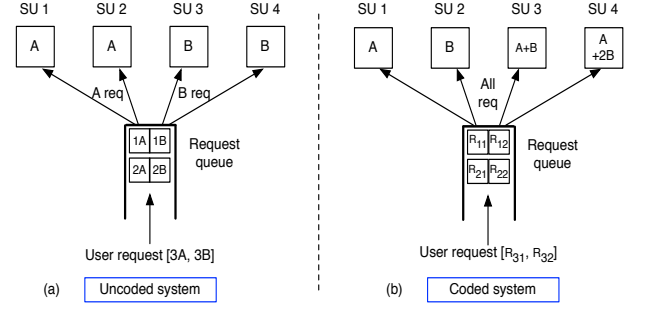

Fig. 2. An example storage system with 4 storage units (SU). The letter inside the SU box corresponds to the packet it stores. Each request arrival brings two packet requests into the system, which must be served by 2 SUs with distinct packets. A single queue is maintained for all the requests. (a) shows the uncoded system, where the packet A requests are served by SU 1 and 2, and the packet B requests are served by SU 3 and 4. (b) shows the coded system, where any two distinct SUs can serve the both packet requests A and B.

served. With such a long memory, even defining an appropriate system state is very hard, let alone analyzing it. To make the analysis tractable, we focus on a small-sized problem with a single content, 2 packets, and n servers. We provide a theoretical upper bound on the average content retrieval delay performance of the coded system, which we show still beats that of a replication-based system with identical amount of SU resources. We believe that the methodology and algorithms developed in this paper will provide useful insights to more general cases, which are part of our ongoing work.

III. STORAGE AND SCHEDULING SCHEMES

In this section, we present the storage and scheduling schemes for both uncoded and coded systems. We assume that *both systems have identical resources*: each system hosts a single content that is divided into $k = 2$ packets, and has $n = 2r, r \geq 2$ SUs, each capable of storing 1 packet. Each server can serve requests with rate of $\mu = 1$. Here r is a redundancy factor in the system for both content availability as well as reliability of the content.

A. Uncoded system

In this case, due to inherent symmetry of arrivals of sub-requests for packets A, B, r SUs store packet A and the remaining r store packet B. Then, whenever there is an idle SU that stores packet A, the packet A request from the head-of-line request is assigned to this server. The same happens for packet B requests.

Under this setting, the uncoded system can be modeled as two $M/M/r$ queueing systems (See Fig. 2). [2] Now denote by π_i the steady-state probability that there are i packet requests in an $M/M/r$ system, and denote $\rho \triangleq \frac{\lambda}{r\mu}$. We note that $\{\pi_0, \pi_1, \ldots, \}$ can be computed as follows [2].

$$\pi_0 = \left(\sum_{m=0}^{r-1} \frac{(r\rho)^m}{m!} + \frac{(r\rho)^r}{r!} \cdot \frac{1}{1-\rho} \right)^{-1}, \quad (1)$$

$$\pi_m = \frac{(r\rho)^m}{m!} \pi_0, \ \forall \, m \in \{1, \ldots, r\}, \quad (2)$$

---

[1]Note that such shared queue models have also been widely used to study data center problems, e.g.,[7] [6].

[2]Note that the two separate systems are indeed not independent, since the arrivals to the systems happen at the exact same time. However, this does not affect the analysis for the average request delay.

$$\pi_{r+m} \;\; = \;\; \rho^m \frac{(r\rho)^r}{r!}\pi_0, \;\; \forall\, m \geq 1. \tag{3}$$

And the average packet request delay $d_{\text{packet}}^{\text{uncoded}}$ can be computed by:

$$d_{\text{packet}}^{\text{uncoded}} = \frac{\sum_{n=0}^{\infty} n\pi_n}{\lambda}. \tag{4}$$

Although the uncoded system admits an easy analysis of the average packet request delay, we see that finding the average request delay can be quite challenging. However, as we will see in the coded system, the average request delay can be easily computed under our algorithm.

B. Coded system: Blocking-One Scheduling (BoS)

We now specify our coding and scheduling schemes for the coded system. We have $n = 2r$ SUs and $k = 2$ packets A, B. We adopt a simple linear $(n, 2)$ MDS code (any family of MDS code will do) to generate n encoded packets that are stored at each of the SU. Due to the MDS nature of the code, any request that is served at *any two distinct SUs will be able to retrieve the full content*, e.g., see Fig. 2(b). Under such a coding scheme, in order to minimize average packet request delay, a simple greedy scheduling strategy would be: queue all the requests in a single queue. Whenever there is an idle SU that can serve any request in the queue, assign the request to that SU. Although the suggested greedy scheme seems simple, it has inherent memory/dependency in scheduling the requests due to the use of codes. For example, if the first packet request of the ith request, denoted by R_{i1}, is served at SU j, then the greedy scheduler has to remember not to assign the other packet request R_{i2} to SU j even if it becomes idle. This dependency builds infinite memory into the system, which makes it very challenging to exactly analyze the average delay performance of requests in the coded system.

In order to resolve this difficulty, we propose a novel scheduler called Blocking-one Scheduling (BoS), for the coded system. The main idea of BoS is to break the memory in the scheduler by blocking the requests beyond the head-of-line request until both the packet requests of the head-of-line request are served. The BoS scheduler also corresponds to first-come-first-serve (FCFS). As we will see, the BoS algorithm not only greatly simplifies the analysis of the packet request delay, but also allows us to directly calculate the average request delay.

Algorithm 1 Blocking-one Scheduling (BoS)

1: At any time t, denote the set of idle SUs as $\mathcal{S}_{\text{Idle}}$, do:

   1) If $\mathcal{S}_{\text{Idle}} \neq \phi$, assign the packet requests from the head-of-line request to an idle SU in $\mathcal{S}_{\text{Idle}}$ as follows:

      • If packet request 1 has not yet been assigned, assign it to the idle SU.

      • Else assign packet request 2 if corresponding packet request 1 was not served by the idle server.

   2) If a packet request is assigned to the SU, change the state of SU from idle to busy and remove it from $\mathcal{S}_{\text{Idle}}$, and remove the packet request from queue.

   3) Repeat step 1) until no further assignment can be made.

Note that under BoS, it can happen that there exists an idle SU but no assignment is made even when the queue is non-empty. For example, when the free SU has served the packet request 1 of the head-of-line request and no other SU is idle. In this case, the requests beyond the head-of-line request are "blocked." Due to this blocking effect, there will be a throughput loss due to the lost scheduling opportunity. Fortunately, as we will see later, the fraction of throughput loss is $O(\frac{1}{r^2})$ that goes to zero as the r increases. Indeed, even for $r = 2$, the BoS achieves 96% of the maximum system throughput.

IV. ANALYZING THE BoS ALGORITHM

We now analyze the BoS algorithm by finding the steady-state distribution of the coded system under BoS. The approach works as follows. We first derive the continuous-time Markov chain that captures the system evolution. Then, we analyze the Markov chain by carefully choosing a set of global balance equations that allow us to compute the steady-state distribution.

A. The system evolution

In this section, we present the Markov chain that models the system evolution. Towards that end, we first take a closer look at the state evolution of the example system in Fig.2 (b) with 4 SUs. Suppose the system is in a state as shown in Fig. 4 (a). In
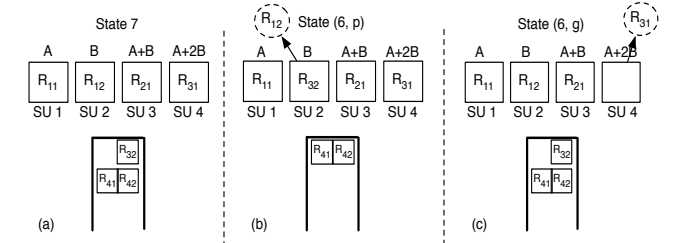

Fig. 4. System evolution of an example with 4 storage units (SU). (a) shows a state with 7 packet requests in the system with an aggregate service rate of 4μ. (b) shows the resulting state after the departure of R_{12} from SU 2, denoted by $(6, p)$, where the system "renews" itself and every queued requests can go to any SU. (c) shows the resulting state after the departure of R_{31} from SU 4, denoted by $(6, g)$. In this case, one SU is "wasted" due to blocking, and the system serves requests with rate 3μ.

this case, the 4 SUs are serving packet requests R_{11}, R_{12}, R_{21}, and R_{31}. There are three more packet requests R_{32}, R_{41} and R_{42} in the queue. Thus, the total number of packet requests in the system is 7. We denote the state of the system by the total number of packet requests in the system i.e., state is 7.

Now, if SU 2 completes its service, the packet request R_{32} can be assigned to SU 2. This results in a state $(6, p)$ as shown in the Fig. 4 (b), which we refer to as a "perfect-state." This state is called "perfect" because once in this state, past evolution is irrelevant in determining the future scheduling events and the total service rate is maximum i.e., 4μ. Note that the system will also enter $(6, p)$ if either SU 1 or 3 completes its service. Hence, the transition rate from state 7 into $(6, p)$ is 3μ. However, if starting from state 7 but SU 4 completes the service, then we cannot assign request R_{32} to SU 4, since R_{31} was served at SU 4. In this case BoS will block the
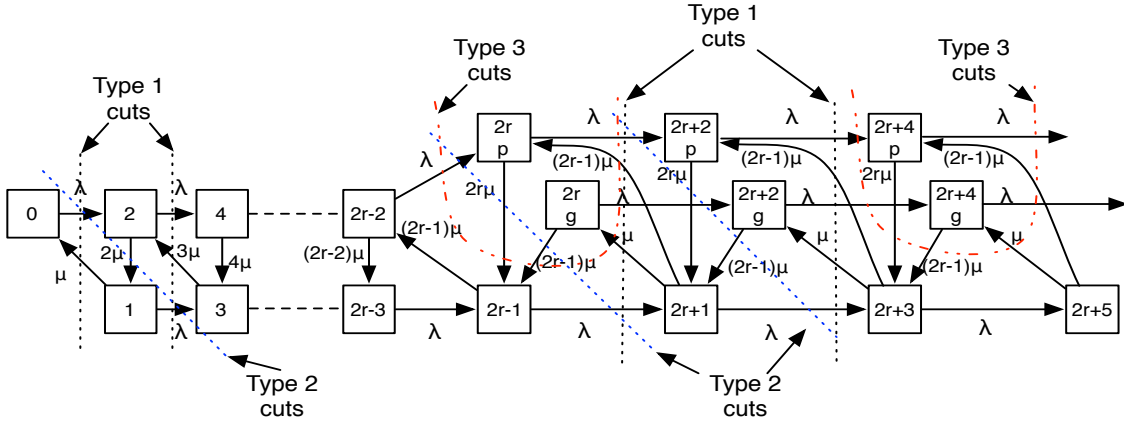
Fig. 3. The continuous-time Markov chain of the system with $2r$ storage units. The request arrival rate is λ and the service rate of each storage unit is μ. Each state of the chain represents the number of packet requests in the system. The letters "p" and "g" represent the perfect and good states. The cuts are used to specify the global balance equations we will use in our analysis.

requests R_{41}, R_{42} until R_{32} gets assigned to some other SU. This results in a not so perfect state, which we denote by $(6, g)$, i.e., good state with 6 packet requests in the system (see Fig.4 (c)). In this good state, the aggregate service rate is 3μ. Note that different from $(6, p)$, the system enters $(6, g)$ from state 7 only when SU 4 finishes its service. Hence, this transition happens only with rate μ.

Although BoS performs sub-optimally as compared to a system with codes that maintains an infinite memory and assigns R_{41} to SU 4 when it becomes idle, it permits analytical treatment of average request delay. It can be verified that this blocking situation appears only when the system is in a state that has more than $2r$ packet requests in the system, and the number of packet requests is even. Thus, to characterize the system state, we introduce the suffix "p" and "g," which stand for "perfect" and "good," for all the states with $2r+2m, m \geq 0$ packet requests in the system. When the number of packet requests in the system is less than $2r$, we simply use the total number of packet requests in the system to denote the state of the system.

Now that the system states have been defined, the Markov chain in Fig. 3 explains the evolution of the system under BoS. The chain can be understood as follows:

- With rate λ, there is an request arrival event, and two new packet requests are added to the system. If system was in a good state before arrival it remains in the good state after arrival and likewise for a perfect state.
- For any state $< 2r$ transition rate for service completion is equal to μ times the state.
- For any odd state $> 2r$ there are two outgoing transitions for service completion:
  1) With rate $(2r - 1)\mu$, there is transition to an even perfect state.
  2) With rate μ, there is transition to an even good state.
- From an even-perfect state $\geq 2r$, there is a service completion transition to an odd state with rate $2r\mu$.
- From an even-good state $\geq 2r$, there is a service completion transition to an odd state with rate $(2r - 1)\mu$.

B. Performance of BoS: Average packet request delay

Here we present the performance results of BoS. To state the theorem, we first define a few notations:

$$\eta \triangleq \frac{\lambda}{2r\mu} + \frac{\lambda(2r-1)\mu}{(2r\mu)^2} + \frac{\lambda\mu}{(2r-1)\mu 2r\mu},$$

$$\gamma_p \triangleq \frac{2r\mu(\lambda + (2r-1)\mu)}{\mu} + (\lambda + 2r\mu),$$

$$\gamma_g \triangleq \frac{-(2r-1)\mu(\lambda + 2r\mu)}{2r\mu} - (2r-1)(\lambda + (2r-1)\mu),$$

$$\beta_p \triangleq \frac{\lambda(\lambda + (2r-1)\mu)}{\mu}, \quad \beta_g \triangleq \frac{-\lambda(\lambda + 2r\mu)}{2r\mu}.$$

Also, for $l \in \{0, ..., 2r - 1\}$, define a_l as follows:

$$a_0 = 1, a_1 = \frac{\lambda}{\mu}a_0, a_l = \frac{\lambda}{l\mu}(a_{l-1} + a_{l-2}). \tag{5}$$

Now denote by $\pi_0, ..., \pi_{2r-1}, \pi_{2r}^p, \pi_{2r}^g, \pi_{2r+1}, ...$ the stationary distribution of the Markov chain in Fig. 3, where superscripts "p" and "g" stand for perfect and good states. We now have the following theorem.

Theorem 1: Under BoS, we have the following:

(a) The maximum rate the coded system can support is:

$$0 \leq \lambda < r\mu\left(1 - \frac{1}{8r^2 - 4r + 1}\right). \tag{6}$$

(b) If the system is stable, i.e., (6) is satisfied, the steady state probabilities can be computed by the following iterative process:

$$\pi_0 = \frac{1 - \eta}{(1 - \eta)\sum_{l=0}^{2r-2} a_l + \frac{\lambda a_{2r-2}}{2r\mu} + a_{2r-1}},$$

$$\pi_l = a_l\pi_0, \forall l \in \{1, ..., 2r - 1\},$$

$$\pi_{2r+2m-1} = \frac{\lambda}{2r\mu}(\pi_{2r+2m-2}^p + \pi_{2r+2m-2}^g + \pi_{2r+2m-3}),$$

$$\pi_{2r}^p = \frac{1}{\gamma_p}\left[\beta_p(\pi_{2r-1} + \pi_{2r-2}) + \lambda\pi_{2r-2}\right]$$

$$\pi_{2r+2m}^p = \frac{1}{\gamma_p}\left[\beta_p(\pi_{2r+2m-2}^p + \pi_{2r+2m-2}^g + \pi_{2r+2m-1}) \right.$$
$$\left. + \lambda\pi_{2r+2m-2}^p - (2r-1)\lambda\pi_{2r+2m-2}^g\right], \forall m \geq 1.$$

π_{2r}^g and π_{2r+2m}^g can be similarly computed by replacing γ_p, β_p with γ_g, β_g. $\diamond$

Proof: See Appendix A. ■

Note that a system with $2r$ servers should ideally support any arrival rate $0 \leq \lambda < r\mu$. However, we see from equation (6) of Theorem 1 that, under BoS, the there is a loss in throughput of order $O(\frac{1}{r^2})$. This loss quickly goes to zero as r increases. Thus, BoS indeed ensures high throughput of the coded system even for moderate values of r. Part (b) of Theorem 1 then provides an efficient way for analyzing the system. We also note that our results are not asymptotic and can be applied to systems of small sizes.

Using the approach in Theorem 1, we can compute the average packet request delay in the system using the equation (4). We then compute the delay gain of coding by:

$$\text{Delay Gain} = (d_{\text{packet}}^{\text{uncoded}} - d_{\text{packet}}^{\text{coded}})/d_{\text{packet}}^{\text{uncoded}}. \tag{7}$$

Fig. 5 shows the delay gain for different values of r. We see that the gain is significant even for small r, e.g., gain is 13% for $r = 4$, and can be up to 17% when $r = 10$. Finally, we emphasize that Fig. 5 is obtained *analytically*, computed using results in Theorem 1. We note that in Fig. 5, for every r value, delay gain is only plotted for arrivals that are within the capacity region of the system, i.e., (6).
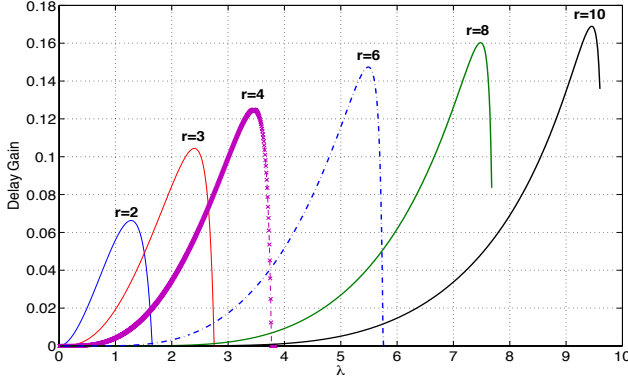

Fig. 5. Delay reduction of BoS over the replication scheme. When $r = 4$, the delay gain is around 13%, and when $r = 10$, the gain is 17%. The plots are generated using analytical results in Theorem 1. The reason the gain decreases as the arrival rate increases is because BoS has a slightly smaller capacity region. Thus, if the rate is very close to the capacity region of BoS, the delay under BoS will be larger than that under the replication scheme. However, this happens only when the rate is very close to the capacity boundary. For most of the rates, codes achieve a significant delay reduction.

C. Average request delay

Notice that the above analysis allows us to derive the average *packet request* delay. However, in practice, we care more about the average *request* delay, which is defined as the average time it takes for both packet requests of a request to get served. Our following lemma shows that under BoS, the average request delay is roughly equal to the packet delay. This is a very desired feature not possessed by the uncoded system.

Lemma 1: Let $d_{\text{req}}^{\text{coded}}$ and $d_{\text{packet}}^{\text{coded}}$ be the average request delay and average packet request delay under BoS, then:

$$d_{\text{req}}^{\text{coded}} = d_{\text{packet}}^{\text{coded}} + \frac{2r-1}{2r-2}\frac{1}{2\mu} \tag{8}$$
$$- \frac{1}{(2r-2)(2r-1)2r\mu} - \frac{1}{2(2r-1)\mu}. \diamond$$

Proof: See Appendix B. ∎

We see from the lemma that as number of servers $2r$ gets large, the difference between the packet delay and the request

delay roughly equals $\frac{1}{2\mu}$. This is exactly the difference between the average of two requests service times and the maximum of them. Hence, it will appear under any scheduling policies regardless of using coding or not. Therefore, Lemma 1 shows that BoS ensures that the average packet latency is almost equal to the average request latency. This is a very important feature of the BoS algorithm.

V. Conclusion

In this paper, we pose a fundamental question of the role of codes in improving the latency of content retrieval in storage systems. The interplay between coding and queueing delay is of complex nature. As a first step to make progress on this complex problem we propose and analyze a simplified setting of a single content divided into two parts and served by multiple servers. We see that even in this simplified setting the exact analysis of queueing delay for the systems using codes is intractable. As a result we provide a sub-optimal scheduling algorithm called Blocking-one Scheduling (BoS) that allows us to theoretically quantify the gains in latency achieved by coded system as compared to a system that uses replication. The methodology we developed in this paper is applicable to a more general setting that allows splitting the content into more than 2 parts. Further generalizations of our work a) extending our scheduling algorithm to incorporate more than one blocking to improve the analytical gains in latency, and b) considering scenario of serving multiple contents, is part of our future work.

Appendix A – Proof of Theorem 1

We now proof Theorem 1 by analyzing the Markov chain using a set of carefully chosen global balance equations, which we call "cuts." Our approach is to first compute π_0. Then starting from π_0, we iteratively compute all the other probabilities.

Proof: (Theorem 1) First consider the sets that contain the $(2r + 2m, p)$ and $(2r + 2m, g)$ states, i.e., the Type 3 cuts in Fig. 3. We note that if the system is stable, then the total transition rate going out from any set of states must be equal to the total rate going into them. Thus, we first have the following equation for the states $(2r, p)$ and $(2r, g)$:

$$\pi_{2r}^p(\lambda + 2r\mu) + \pi_{2r}^g(\lambda + (2r - 1)\mu) \tag{9}$$
$$= \pi_{2r-2}\lambda + \pi_{2r+1}2r\mu.$$

Then for states $(2r + 2m, p)$ and $(2r + 2m, g)$ with $m \geq 1$, we have:

$$\pi_{2r+2m}^p(\lambda + 2r\mu) + \pi_{2r+2m}^g(\lambda + (2r - 1)\mu) \tag{10}$$
$$= \lambda(\pi_{2r+2m-2}^p + \pi_{2r+2m-2}^g) + \pi_{2r+2m+1}2r\mu.$$

Summing (9) and (10) over $m = 1, 2, ...,$ we get:

$$2r\mu \sum_{m=0}^{\infty} \pi_{2r+2m}^p + (2r - 1)\mu \sum_{m=0}^{\infty} \pi_{2r+2m}^g$$
$$= \lambda\pi_{2r-2} + 2r\mu \sum_{m=0}^{\infty} \pi_{2r+2m+1}. \tag{11}$$

Now consider the diagonal cuts, i.e., the Type 2 cuts, starting from states $(2r, p)$ and $(2r, g)$. We have:

$$(\pi_{2r-2} + \pi_{2r-1})\lambda = \pi_{2r}^p 2r\mu + \pi_{2r}^g(2r - 1)\mu, \tag{12}$$

$$(\pi_{2r+2m}^p + \pi_{2r+2m}^g + \pi_{2r+2m+1})\lambda \tag{13}$$
$$= \pi_{2r+2m+2}^p 2r\mu + \pi_{2r+2m+2}^g(2r-1)\mu, \ m \geq 0.$$

Summing (12) and (13) over $m = 0, 1, ...,$ we get:

$$2r\mu \sum_{m=0}^{\infty} \pi_{2r+2m}^p + (2r-1)\mu \sum_{m=0}^{\infty} \pi_{2r+2m}^g$$
$$= \lambda\Big[1 - \sum_{l=0}^{2r-3} \pi_l\Big]. \tag{14}$$

Using (11) and (14), we thus obtain:

$$2r\mu \sum_{m=0}^{\infty} \pi_{2r+2m+1} = \lambda\Big[1 - \sum_{l=0}^{2r-2} \pi_l\Big]. \tag{15}$$

We now try to first find π_0. Consider only the "p" states, we get:

$$\pi_{2r}^p(\lambda + 2r\mu) = \pi_{2r-2}\lambda + \pi_{2r+1}(2r-1)\mu, \tag{16}$$
$$\pi_{2r+2m}^p(\lambda + 2r\mu) = \pi_{2r+2m-2}^p\lambda + \pi_{2r+2m+1}(2r-1)\mu. \tag{17}$$

Summing (16) and (17) over $m \geq 1$, we obtain:

$$2r\mu \sum_{m=0}^{\infty} \pi_{2r+2m}^p = \lambda\pi_{2r-2} + (2r-1)\mu \sum_{m=0}^{\infty} \pi_{2r+2m+1}. \tag{18}$$

Similarly, we can look at the "g" states and get:

$$\pi_{2r}^g(\lambda + (2r-1)\mu) = \pi_{2r+1}\mu, \tag{19}$$
$$\pi_{2r+2m}^g(\lambda + (2r-1)\mu) = \pi_{2r+2m-2}^g\lambda + \pi_{2r+2m+1}\mu. \tag{20}$$

Summing these up, we get:

$$(2r-1)\mu \sum_{m=0}^{\infty} \pi_{2r+2m}^g = \mu \sum_{m=0}^{\infty} \pi_{2r+2m+1}. \tag{21}$$

Using (15), (18) and (21), we obtain:

$$\sum_{m=0}^{\infty} \Big[\pi_{2r+2m}^p + \pi_{2r+2m}^g + \pi_{2r+2m+1}\Big]$$
$$= \Big[\frac{\lambda}{2r\mu} + \frac{\lambda(2r-1)\mu}{(2r\mu)^2} + \frac{\lambda\mu}{(2r-1)\mu 2r\mu}\Big]\Big[1 - \sum_{l=0}^{2r-2}\pi_l\Big]$$
$$+ \frac{\lambda}{2r\mu}\pi_{2r-2}.$$

Therefore, we get:

$$\Big[\frac{\lambda}{2r\mu} + \frac{\lambda(2r-1)\mu}{(2r\mu)^2} + \frac{\lambda\mu}{(2r-1)\mu 2r\mu}\Big]\Big[1 - \sum_{l=0}^{2r-2}\pi_l\Big] \tag{22}$$
$$+ \frac{\lambda}{2r\mu}\pi_{2r-2} = \Big[1 - \sum_{l=0}^{2r-2}\pi_l\Big] - \pi_{2r-1}.$$

We see that (22) provides one equation in terms of only $\pi_0, ..., \pi_{2r-1}$. Below we show that all the probabilities $\pi_1, ..., \pi_{2r-1}$ can be expressed in terms of π_0. In this case (22) will allow us to compute π_0 exactly. This in turn enables us to compute $\pi_1, ..., \pi_{2r-1}$. To do so, we first consider the type 1 and type 2 cuts shown in Fig. 3 to get:

$$\pi_1 = \frac{\lambda}{\mu}\pi_0, \tag{23}$$
$$\pi_{2i} = \frac{\lambda}{2i\mu}(\pi_{2i-2} + \pi_{2i-1}), \ \forall \, i \in \{1, ..., r-1\}, \tag{24}$$
$$\pi_{2i+1} = \frac{\lambda}{(2i+1)\mu}(\pi_{2i-1} + \pi_{2i}), \forall \, i \in \{1, ..., r-1\}. \tag{25}$$

Using (23), (24) and (25), one can obtain:

$$\pi_l = a_l\pi_0, \ \forall \, l \in \{0, ..., 2r-1\}, \tag{26}$$

where $\{a_l, l = 0, ..., 2r-1\}$ are defined as in (5). Plugging (26) back into (22) and denote $\eta \triangleq \frac{\lambda}{2r\mu} + \frac{\lambda(2r-1)\mu}{(2r\mu)^2} + \frac{\lambda\mu}{(2r-1)\mu 2r\mu}$, we have:

$$1 - \sum_{l=0}^{2r-2} a_l\pi_0 = \eta - \eta\sum_{l=0}^{2r-2} a_l\pi_0 + \frac{\lambda}{2r\mu}a_{2r-2}\pi_0 + a_{2r-1}\pi_0.$$

Therefore:

$$\pi_0 = \frac{1 - \eta}{(1-\eta)\sum_{l=0}^{2r-2} a_l + \frac{\lambda a_{2r-2}}{2r\mu} + a_{2r-1}}. \tag{27}$$

It is not difficult to verify that π_0 is a valid probability if $1 - \eta > 0$, i.e.,

$$\frac{\lambda}{2r\mu} + \frac{\lambda(2r-1)\mu}{(2r\mu)^2} + \frac{\lambda\mu}{(2r-1)\mu 2r\mu} < 1. \tag{28}$$

This implies that the supportable rate is:

$$\lambda < r\mu\frac{1 - \frac{1}{2r}}{1 - \frac{1}{2r} + \frac{1}{8r^2}} = r\mu\Big(1 - \frac{1}{8r^2 - 4r + 1}\Big). \tag{29}$$

Here $r\mu$ is the total rate the system can ever support. Hence, we see that BoS only lose a fraction $\frac{1}{8r^2-4r+1}$. This proves Part (a).

To prove Part (b), we first see that one can now use (26) to compute $\pi_0, ..., \pi_{2r-1}$. To compute $\pi_{2r+2m}^p, \pi_{2r+2m}^g$, for all $m \geq 0$, we start from $m = 0$. We have from (12) that:

$$\pi_{2r}^p 2r\mu + \pi_{2r}^g(2r-1)\mu = \lambda(\pi_{2r-1} + \pi_{2r-2}). \tag{30}$$

Now if we look at the state $(2r, p)$ and $(2r, g)$ separately, we get:

$$\pi_{2r}^p(\lambda + 2r\mu) = \pi_{2r-2}\lambda + \pi_{2r+1}(2r-1)\mu, \tag{31}$$
$$\pi_{2r}^g(\lambda + (2r-1)\mu) = \pi_{2r+1}\mu. \tag{32}$$

Canceling the term π_{2r+1} in (31) and (32), we get that:

$$\pi_{2r}^p(\lambda + 2r\mu) - \pi_{2r}^g(2r-1)(\lambda + (2r-1)\mu) = \lambda\pi_{2r-2}. \tag{33}$$

With (30) and (33), we can now compute π_{2r}^p, π_{2r}^g. To make the expressions more concise, we define:

$$\gamma_p \triangleq \frac{2r\mu(\lambda + (2r-1)\mu)}{\mu} + (\lambda + 2r\mu),$$
$$\gamma_g \triangleq \frac{-(2r-1)\mu(\lambda + 2r\mu)}{2r\mu} - (2r-1)(\lambda + (2r-1)\mu),$$
$$\beta_p \triangleq \frac{\lambda(\lambda + (2r-1)\mu)}{\mu}, \quad \beta_g \triangleq \frac{-\lambda(\lambda + 2r\mu)}{2r\mu}.$$

Then we get:

$$\pi_{2r}^p = \frac{1}{\gamma_p}\Big[\beta_p(\pi_{2r-1} + \pi_{2r-2}) + \lambda\pi_{2r-2}\Big], \tag{34}$$
$$\pi_{2r}^g = \frac{1}{\gamma_g}\Big[\beta_g(\pi_{2r-1} + \pi_{2r-2}) + \lambda\pi_{2r-2}\Big]. \tag{35}$$

Now for all the states $(2r + 2m, p)$ and $(2r + 2m, g)$ with $m \geq 1$, using (13), (17) and (20), we get:

$$2r\mu\pi_{2r+2m}^p + (2r-1)\mu\pi_{2r+2m}^g$$
$$= \lambda(\pi_{2r+2m-2}^p + \pi_{2r+2m-2}^g + \pi_{2r+2m-1}),$$
$$(\lambda + 2r\mu)\pi_{2r+2m}^p - (2r-1)(\lambda + (2r-1)\mu)\pi_{2r+2m}^g$$
$$= \lambda\pi_{2r+2m-2}^p - (2r-1)\lambda\pi_{2r+2m-2}^g.$$

We can thus obtain the following equations for all states $2r + 2m, m \geq 1$: [3]

$$\pi_{2r+2m}^p = \frac{1}{\gamma_p}\Big[\beta_p(\pi_{2r+2m-2}^p + \pi_{2r+2m-2}^g + \pi_{2r+2m-1})$$

---

[3] It can be verified that both π_{2r+2m}^p and π_{2r+2m}^g are both positive. Thus they are valid probabilities.

$$+\lambda\pi^p_{2r+2m-2} - (2r-1)\lambda\pi^g_{2r+2m-2}\big],$$
$$\pi^g_{2r+2m} = \frac{1}{\gamma_g}\big[\beta_g(\pi^p_{2r+2m-2} + \pi^g_{2r+2m-2} + \pi_{2r+2m-1})$$
$$+\lambda\pi^p_{2r+2m-2} - (2r-1)\lambda\pi^g_{2r+2m-2}\big].$$

Then, the probabilities $\pi_{2r+2m-1}$ with $m \geq 1$ can be computed using type 1 cuts, i.e.,

$$2r\mu\pi_{2r+2m-1} = \lambda(\pi^p_{2r+2m-2} + \pi^g_{2r+2m-2} + \pi_{2r+2m-3}). \quad (36)$$

With all the above results, the average packet request delay in the system can be computed as:

$$d^{\text{coded}}_{\text{packet}} = \frac{1}{2\lambda}\Bigg[\sum_{l=1}^{2r-1} l\pi_l + \sum_{m\geq 0}(2r+2m+1)\pi_{2r+2m+1}$$
$$+ \sum_{m\geq 0}(2r+2m)(\pi^p_{2r+2m} + \pi^g_{2r+2m})\Bigg].$$

This completes the proof of Part (b). ∎

APPENDIX B – PROOF OF LEMMA 1

Here we prove Lemma 1.

Proof: (Lemma 1) Consider any request u that enters and departs from the system. Let w_{u1} and w_{u2} be the waiting times of its first and second packet requests in the queue. Then let s_{u1} and s_{u2} be the service times of the two packet requests.

We now derive a relationship between w_{u1} and w_{u2}. According to BoS, packet request 1 always goes before packet request 2, thus $w_{u1} \leq w_{u2}$. Now suppose packet request 1 goes to a storage unit j at a time t. Then, the extra waiting time of packet request 2 in the queue is exactly the time it takes for any of the other $2r-1$ storage units to be free. By the exponential service time nature, this time is exponentially distributed with rate $(2r-1)\mu$, i.e.,

$$w_{u2} = w_{u1} + \tau_u, \quad \text{where } \tau_u \sim \exp((2r-1)\mu). \quad (37)$$

Now let t_{u1} and t_{u2} be the total times packet request 1 and 2 stay in the system, and let t_u be the time the request spends in the system, we have:

$$t_{u1} = w_{u1} + s_{u1}, \quad (38)$$
$$t_{u2} = w_{u2} + \tau_u + s_{u2}, \quad (39)$$
$$t_u = w_{u1} + \max(s_{u1}, s_{u2} + \tau_u). \quad (40)$$

Denote $Z = \max(s_{u1}, s_{u2} + \tau_u)$. It can be verified that:

$$f_Z(z) = \frac{2r-1}{2r-2}\mu e^{-\mu z} - \frac{(2r-1)\mu}{2r-2}e^{-(2r-1)\mu z}$$
$$+ \mu e^{-\mu z} - \frac{2r-1}{2r-2}2\mu e^{-2\mu z} + \frac{1}{2r-2}\mu e^{-2r\mu z},$$

and:

$$\mathbb{E}\Big[Z\Big] = \frac{1}{\mu} + \frac{2r-1}{2r-2}\frac{1}{2\mu} - \frac{1}{(2r-2)(2r-1)2r\mu}. \quad (41)$$

It thus follows that the difference between the average request delay and the average packet request delay under BoS is given by:

$$d^{\text{coded}}_{\text{req}} - d^{\text{coded}}_{\text{packet}} = \mathbb{E}\Big[t_u\Big] - \frac{1}{2}\Big(\mathbb{E}\Big[t_{u1}\Big] + \mathbb{E}\Big[t_{u2}\Big]\Big)$$
$$= \frac{2r-1}{2r-2}\frac{1}{2\mu} - \frac{1}{(2r-2)(2r-1)2r\mu} - \frac{1}{2(2r-1)\mu}.$$

This completes the proof. ∎

REFERENCES

[1] G. Ananthanarayanan, S. Agarwal, S. Kandula, A Greenberg, and I. Stoica. Scarlett: Coping with skewed content popularity in mapreduce clusters. *Proceedings of ACM EuroSys*, 2011.
[2] D. P. Bertsekas and R. G. Gallager. *Data Networks (2nd Edition)*. 1992.
[3] M. Bramson, Y. Lu, and B. Prabhakar. Randomized load balancing with general service time distributions. *Proceedings of ACM Sigmetrics*, 2010.
[4] T. Dikaliotis, A. G. Dimakis, T. Ho, and M. Effros. On the delay of network coding over line networks. *IEEE International Symposium on Information Theory (ISIT)*, 2009.
[5] A. Eryilmaz, A. Ozdaglar, M. Medard, and Ebad Ahmed. On the delay and throughput gains of coding in unreliable networks. *IEEE Transactions on Information Theory*, Dec 2008.
[6] A. Gandhia and M. Harchol-Baltera. How data center size impacts the effectiveness of dynamic power management. *Proc. of the Forty-Ninth Annual Allerton Conference*, 2011.
[7] A. Gandhia, M. Harchol-Baltera, and I. Adan. Server farms with setup costs. *Performance Evaluation,Volume 67 Issue 11*, Nov 2010.
[8] X. Li, C.-C. Wang, and X. Lin. Throughput and delay analysis on uncoded and coded wireless broadcast with hard deadline constraints. *Proceedings of IEEE INFOCOM Mini-Conference*, March 2010.
[9] Y. Lu, Q. Xie, G. Kliot, A. Geller, J. Larus, and A. Greenberg. Join-idle-queue: A novel load balancing algorithm for dynamically scalable web services. *29th International Symposium on Computer Performance, Modeling, Measurements, and Evaluation (Performance)*, 2011.
[10] A. ParandehGheibi, M. Medard, A. Ozdaglar, and S. Shakkottai. Avoiding interruptionsa qoe reliability function for streaming media applications. *to appear in IEEE Journal on Selected Areas in Communications, special issue on Trading rate for Delay at the Transport and Application layers*, 2012.
[11] W. Yeow, A. Hoang, and C. Tham. Minimizing delay for multicast-streaming in wireless networks with network coding. *Proceedings of IEEE INFOCOM*, 2009.