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 use codes 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 introduce 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 erasure codes to introduce redundancy 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 servers, called storage units (SU), 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) replication: each packet is replicated two times; 2) coding: file is encoded using a $(4, 2)$ Maximum-Distance-Separable (MDS) code. It is easy to see that the redundancy factor is 2 in both cases. There is a common dispatcher that queues and schedules the incoming requests. The request process is Poisson and each file request contains two packet-requests for two packet components. The service time of the storage unit is exponentially distributed.

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

Quantifying the exact request delay in a coded system is a challenging task. The main difficulty is that *the optimal scheduling algorithm needs to remember which SUs served earlier requests* to ensure that the two requests of a particular content are 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] [10] are not relevant for this setup.

To tackle this challenge and to demonstrate the role of codes in delay reduction, we develop a novel scheduling algorithm called Blocking-one Scheduling (BoS) to derive a close upper bound of the delay of the coded system. The idea of BoS is to block 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 the system using codes. Under BoS, we analytically show that the system that uses codes reduces the average request delay by $7\% - 17\%$ compared to the system that uses replication, depending upon the replication factor of the files. The intuition behind this is that in the replication system, the requests for packet A or B can be satisfied by specific designated 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.

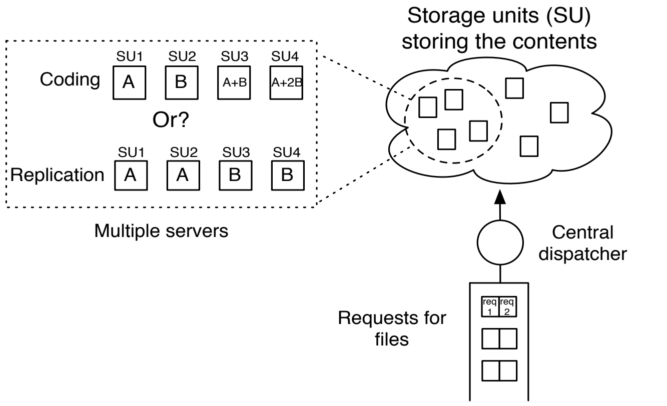

Fig. 1. An example information storage system storing a file that consists of two packets A and B. Requests for contents are assigned by a central dispatcher to available storage units. In order to reduce the content retrieval latency, we can either use *replication*, i.e., store the packets A, B twice,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

Related Work: The authors in [5] and [12] study the throughput-delay gains of network coding in a single hop wireless downlink with unreliable channels. The authors of [9] consider the network coding gains in throughput when packets have hard delay deadlines. The work of [4] studies the gains in delay and throughput when using network coding over a linear network with unreliable links. In [11], 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 (SU). Each SU has a fixed storage capacity, and is capable of serving each incoming 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 *content-request* into k *packet-requests* and by making sure that no two packet-requests of a content-request 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 further illustrate the problem, 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. 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

[1]In practice, each server usually maintains a separate queue. However, due to the use of dispatchers, the queues typically remain short. Thus, such shared queue models have also been used to study server farm systems in data centers, e.g., [7] [6].

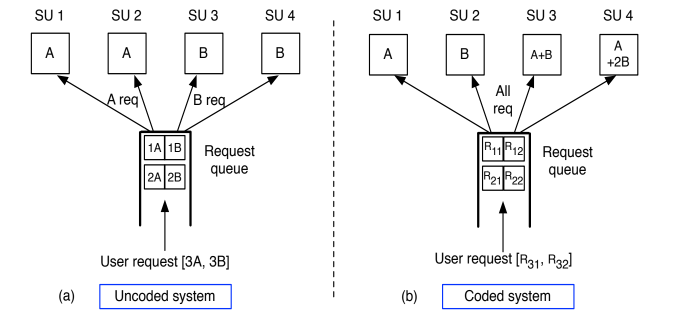

Fig. 2. An example storage system with 4 storage units (SU). The letter inside the SU box corresponds to the packet it stores. Each request arrival brings two packet requests into the system, which must be served by 2 SUs with distinct packets. A single queue is maintained for all the requests. (a) shows 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.

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 served. With such a long memory in scheduling, even defining an appropriate system state is very hard, let alone analyzing it.

To make the analysis tractable, we focus on a special case with a single content, 2 packets, and n servers. We provide a close theoretical upper bound on the average content retrieval delay performance of the coded system, which we show still performs better than 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 is 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, and λ is the arrival rate of the content requests.

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 (a)). [2] Now denote by π_i the steady-state probability that there are i packet

[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 packet request delay.

requests in an $M/M/r$ system, and denote $\rho \triangleq \frac{\lambda}{r\mu}$. We note that $\{\pi_0, \pi_1, \dots,\}$ can be computed as follows [2].

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

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

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

The average packet request delay $d^{\text{uncoded}}_{\text{packet}}$ can be computed by:

$$d^{\text{uncoded}}_{\text{packet}} = \frac{\sum_{n=0}^{\infty} n\pi_n}{\lambda}. \qquad (4)$$

Although the uncoded system admits an easy analysis of the average packet request delay, we see that finding the average content request delay can be quite challenging. However, as we will see in the coded system, the average content 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 linear $(n, 2)$ MDS code (any family of MDS code will do) to generate n encoded packets that are stored at each of the SUs. Due to the MDS nature of the code, any content 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, the optimal scheduling strategy will be to assign a request to an SU whenever it is idle and can serve a packet-request in the queue. Although the suggested 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 i^{th} content request, denoted by R_{i1}, is served at SU j, then the scheduler has to remember not to assign the other packet request R_{i2} to SU j even if it becomes idle. This dependency builds large memory into the system, which makes it very challenging to analyze the average delay performance of the requests in the coded system.

In order to resolve this difficulty, we propose a novel scheduling scheme called Blocking-one Scheduling (BoS), for the coded system. The main idea of BoS is to break the memory in the scheduler, i.e., the scheduler only has to remember which SU serves the HOL packet-requests, by blocking the requests beyond the HOL request until both packet requests of the HOL request are assigned to servers. [3] 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 content request delay.

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 content 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. Hence, the delay performance of BoS serves as an upper bound of the optimal scheduling scheme in the coded system. Interestingly, we show that such an opportunity loss under BoS decreases as $O(\frac{1}{r^2})$. For instance, for $r = 2$, the BoS achieves 96% of the maximum system throughput. Thus, BoS's performance approaches the optimal scheme as r increases, providing us with a way to analyze the exact delay performance of a coded system.

[3]Note that in general one can do a "Blocking-k Scheduling," i.e., if the HOL content request is i, then do not allow any packet requests from $i + k$ or after.

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 as follows:
        • If packet request 1 has not yet been assigned, assign it to an idle server.
        • Else assign packet request 2 to an idle server that did not serve packet request 1.
    2) If an idle server is assigned a packet request, change its state from idle to busy. Remove the server from $\mathcal{S}_{\text{Idle}}$ and remove the packet request from the queue.
    3) Repeat step 1) until no further assignment can be made.
```

IV. Analyzing the BoS algorithm

We now analyze the BoS algorithm by finding the steady-state distribution of the number of packet requests in the coded system. 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 the Fig.2 (b) with 4 SUs.

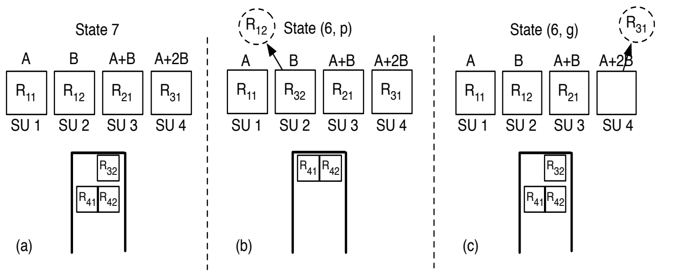

Fig. 4. System evolution of an example with 4 storage units (SU). (a) shows a state with 7 packet requests in the system with an aggregate service rate of 4μ. (b) shows the resulting state after the departure of R_{12} from SU 2, denoted by $(6, p)$, where the system "renews" itself and every queued requests can go to any SU afterwards. (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.

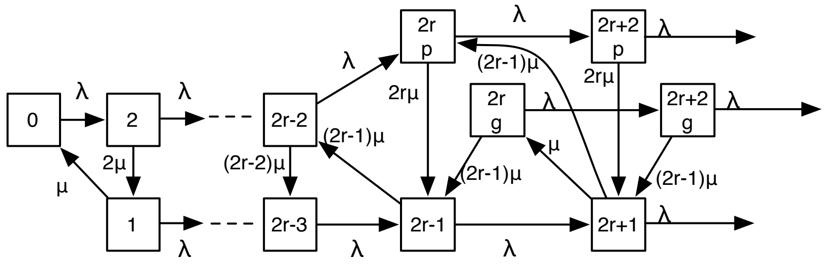

Fig. 3. The continuous-time Markov chain of the system with $2r$ storage units. The request arrival rate is λ and the service rate of each storage unit is μ. Each state of the chain represents the number of packet requests in the system. The letters "p" and "q" represent the perfect and good states.

Consider a system in state as shown in Fig. 4 (a). In 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 this 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. However, if starting from state 7 but SU 4 completes the service first, then we cannot assign request R_{32} to SU 4, since R_{31} was served by SU 4. In this case BoS will block the requests R_{41}, R_{42} until R_{32} gets assigned to some other SU. This results in a not so perfect state hence we denote it by $(6, g)$, i.e., good state with 6 packet requests in the system (see Fig.4 (c)). Note that in this good state the aggregate service rate is only 3μ.

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 has more than $2r$ packet requests, 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" states. When the number of packet requests in the system is less than $2r$ or when it is odd, 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 a request arrival event, and two new packet requests are added to the system. If the system was in a good state before the arrival, it remains in the good state after the 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 a transition to an even perfect state. This corresponds to the event that an SU not serving the HOL request's 1st packet request just completes the service (see Fig. 4 (b)).
  2) With rate μ, there is a transition to an even good state (This corresponds to Fig. 4 (c)).
- 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:

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

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

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

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 have the following theorem:

Theorem 1. *Under BoS, we have the following:*

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

$$0 \leq \lambda < r\mu\left(1 - \frac{1}{8r^2 - 4r + 1}\right). \quad (5)$$

(b) If the system is stable, i.e., (5) is satisfied, the steady state probabilities can be computed by the following iterative process:

$$\pi_0 = \frac{1-\eta}{(1-\eta)\sum_{n=0}^{2r-2} a_l + \frac{\lambda a_{2r-2}}{2r\mu} + a_{2r-1}},$$
$$\pi_l = a_l \pi_0,\ \forall\, l \in \{1, ..., 2r-1\},$$
$$\pi_{2r+2m-1} = \frac{\lambda}{2r\mu}(\pi_{2r+2m-2}^p + \pi_{2r+2m-2}^g + \pi_{2r+2m-3}),$$
$$\pi_{2r}^p = \frac{1}{\gamma_p}\left[\beta_p(\pi_{2r-1} + \pi_{2r-2}) + \lambda\pi_{2r-2}\right],$$
$$\pi_{2r+2m}^p = \frac{1}{\gamma_p}\left[\beta_p(\pi_{2r+2m-2}^p + \pi_{2r+2m-2}^g + \pi_{2r+2m-1})\right.$$

$$+\lambda\pi^{p}_{2r+2m-2} - (2r-1)\lambda\pi^{g}_{2r+2m-2}\big],\ \forall\ m \geq 1.$$

π^g_{2r} and π^g_{2r+2m} can be similarly computed by replacing γ_p, β_p with γ_g, β_g. ◇

Proof: Omitted due to space limit. Please see [8] for the proofs. ■

Note that a system with $2r$ servers should support an arrival rate of $r\mu$. However, we see from equation (5) of Theorem 1 that, under BoS, there is a loss in throughput of $O(\frac{1}{r^2})$. This throughput loss is due to blocking and quickly goes to zero as r increases. Thus, BoS indeed ensures high throughput of the coded system even for moderate values of r, and serves as a good approximation of the optimal scheduling scheme for the coded system. Part (b) of Theorem 1 then provides an efficient way for analyzing the system. We emphasize that *our results are not asymptotic and can be applied to systems of any sizes.*

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

$$\text{Delay Gain} = (d^{\text{uncoded}}_{\text{packet}} - d^{\text{coded}}_{\text{packet}})/d^{\text{uncoded}}_{\text{packet}}. \quad (6)$$

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 $r = 10$. The reason why the gain decreases as λ approaches $r\mu$ is due to the throughput loss of BoS, which leads to a faster increase of delay when $\lambda \to r\mu$. Finally, we emphasize that Fig. 5 is obtained *analytically*, computed using results in Theorem 1.

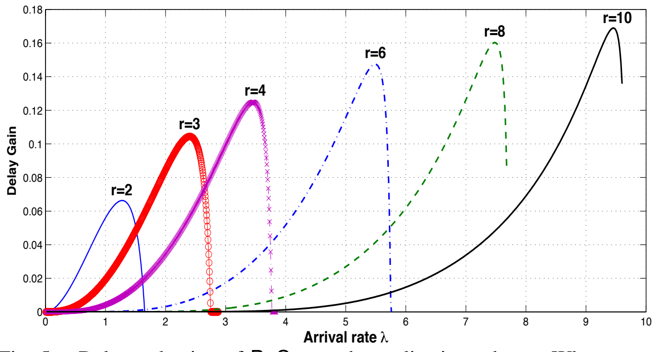

Fig. 5. Delay reduction of BoS over the replication scheme. When $r = 4$, the delay gain is around 13%, and when $r = 10$, the gain is 17%. The plots are generated using analytical results in Theorem 1.

C. Average content request delay

Notice that the above analysis allows us to derive the average *packet request* delay. In practice, we care more about the average *content request* delay, which is defined as the average time it takes for both packet requests of a request to get served. The 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 $d^{coded}_{content}$ and d^{coded}_{packet} be the average content request delay and average packet request delay under BoS, then:*

$$d^{coded}_{content} = d^{coded}_{packet} + \frac{2r-1}{2r-2}\frac{1}{2\mu} - \frac{1}{(2r-2)(2r-1)2r\mu} - \frac{1}{2(2r-1)\mu}. \diamond \quad (7)$$

Proof: Omitted. Please see [8] for the proof. ■

We see from the lemma that as the 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 request 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 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. Our future work will consider scenarios of serving multiple contents.

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] L. Huang, S. Pawar, H. Zhang, and K. Ramchandran. Codes can reduce queueing delay in data centers. *arXiv Technical Report, arXiv:1202.1359v1*, 2012.

[9] 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.

[10] 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.

[11] A. ParandehGheibi, M. Medard, A. Ozdaglar, and S. Shakkottai. Avoiding interruptionsa qoe reliability function for streaming media applications. *to appear in IEEE JSAC, special issue on Trading rate for Delay at the Transport and Application layers*, 2012.

[12] W. Yeow, A. Hoang, and C. Tham. Minimizing delay for multicast-streaming in wireless networks with network coding. *Proceedings of IEEE INFOCOM*, 2009.