Heuristics for Analyzing Download Time in MDS Coded Storage Systems

Mehmet Fatih Aktaş and Emina Soljanin
Department of Electrical and Computer Engineering, Rutgers University
Email: {mehmet.aktas, emina.soljanin}@rutgers.edu

***Abstract*—There has been a growing interest, in both theory and practice, in using the available redundancy in storage systems for mitigating stragglers in content download. This paper is concerned with MDS coded storage systems and studies (n, k) data access model. When $k = n$, system is equivalent to a fork-join queue, which is known to be notoriously hard to analyze, while system with $k = 1$ has been previously shown to be equivalent to an $M/G/1$ queue. We here argue that the system with $k = 2$ is of practical interest, and then present a method that approximates the system as an $M/G/1$ queue. Approximated download time is shown to be more accurate than the bounds available in the literature. We also note that the presented method can be used for approximating systems that employ other newly designed and deployed storage codes.**

I. Introduction

Storage systems distribute and store content with redundancy over multiple nodes. Downloading distributed content requires fetching data from multiple servers in parallel. The probability of a slow download quickly increases with the number of contacted servers due to the server-side variability [1]. Recent research proposes to download by requesting content from both the original and the redundant storage simultaneously such that a set of fastest downloads is sufficient to recover the desired content, while the slow downloads can be avoided and cancelled. For instance, if files are split into k and then encoded into $n > k$ chunks by a Maximum Distance Separable (MDS) code, a file can be recovered as soon as downloading any k of its n chunks.

Load balancing, download *parallelism* and *redundancy* are the three important design choices that affect download time in storage systems. Splitting files into greater number of chunks (greater parallelism) means having to fetch smaller data from each server. However, splitting files into more chunks requires contacting greater number of servers for download, which performs slow with higher probability. Downloading files by simultaneously fetching also the redundant chunks can reduce the probability of a slow download but it introduces additional load on the servers, which may result in higher queueing times.

There has been a growing interest to understand the interplay between load balancing, parallelism and redundancy on download times. Download with replicated redundancy is studied and an exact analysis is given for i.i.d. exponential server service times in [2]. The adopted system model is that each download request is copied into a randomly or deterministically chosen set of servers at arrival and the download is completed as soon as the fastest copy completes. Authors follow up this work in [3] and show that exploiting redundancy at a level beyond a threshold causes further slowdowns in downloads or even instability due to the excessive load introduced on the system by the replicated requests. A model for downloading with MDS coded redundancy is introduced in [4]. Authors name this data access model as (n, k) system, where each arriving request is copied into all n servers and the download is completed as soon as any $k < n$ of them finish. This system is a generalization of fork-join queues, i.e., (n, n) system. Fork-join queues are known to be notoriously hard to analyze and its exact analysis is available only for two-server system under exponential inter-arrival and service times. Approximations exist for their average response time in general [5]. Authors in [6] show that when $k = 1$, system is equivalent to an $M/G/1$ queue and its exact analysis follows from the standard results. For arbitrary value of k, some bounds and approximations for the average download time are presented in [4], [6]–[9]. Recently, new storage codes with unique properties have been designed and deployed in real distributed systems [10]. These codes gave birth to new download schemes with redundancy and consequently new interesting queueing problems for download time analysis [11]–[13].

In this paper, we firstly study the download time in a storage system that distributes download traffic across multiple disjoint sub-systems, where each sub-system implements an independent (n, k) MDS coded storage. We observe that the minimum download time is achieved by employing low levels of parallelism. Note that, low parallelism is also desired in practice to achieve low complexity and overhead in crucial system operations, such as file downloads and recovery from node failures. These motivate us to study download with minimal parallelism $k = 2$. Our observations for arbitrary k allow us to approximate the $(n, 2)$ system as an $M/G/1$ queue, then standard queueing results are used to derive the download time. The presented approximate method gives insight into the system dynamics, and is shown to perform better than the bounds available in the literature in estimating the simulated download time. In addition, the bounds available in the literature cannot be extended for accurate approximation of the systems that employ other recently deployed storage codes (e.g., see [10]). The method given in this paper proved to be accurate for approximating download time in emerging storage systems, e.g., see [13] for an approximate analysis of a Simplex coded storage system.

II. Download from MDS Coded Storage

System Model: Download time is governed by two factors: 1) Queueing time for the download requests, 2) Service time that takes to fetch the data from disk or memory, and stream

it to the user. As real storage systems do, suppose files are split into equal size chunks, e.g., chunks are 64MB in GFS [14], 128MB in HDFS [15]. As commonly adopted in the literature [4], [7], we assume that each file consists of the same number of chunks. This is mainly for tractable analysis and may not hold in practice. We model the service time variability at all servers using the same distribution and assume service times are independent across the servers. All these assumptions together imply that the service time V's are i.i.d. across the chunks and the servers.

We use two canonical service time models: 1) Shifted-Exponential S-Exp(D, μ) where D models the minimum download time due to data size and μ is the decaying rate of its exponential tail modeling the service time variability, 2) Pareto(s, α) with positive minimum value s and a power law tail index α. Pareto is a heavy tailed distribution that is observed to fit service time variability in real systems [1], [16].

We study a fairly tractable and modular storage system model, which we denote as (N, n, k) system. System consists of N servers in total and is divided into N/n (suppose $n|N$) disjoint groups of n servers. Each sub-system stores a different set of files and implements an independent (n, k) coded storage. Within each, files are split into k systematic equal size chunks and expanded with $n - k$ coded chunks, then all of which are distributed across all n servers.

For the sake of tractable analysis, we assume download requests arrive as a Poisson process of rate λ. Requested files are assumed to be found uniformly at random in one of the (n, k) sub-systems, hence each arriving request is forwarded to one of the sub-systems uniformly at random. Thus, the arrivals to each sub-system follow an independent Poisson of rate $\lambda n/N$. Download time for an arbitrary request arriving to (N, n, k) system is then equivalent to the response time of an (n, k) storage system. In each of the (n, k) sub-systems, requests are copied into all n servers and the download is completed as soon as the fastest k of them completes. Remaining $n - k$ redundant copies of a request are removed from the system as soon as its download is completed.

Parallelism, redundancy and load balancing: Important system design choices are determined by k and n as follows: 1) *Parallelism-k*: chunk sizes decrease linearly with the number of chunks k that constitute the files, and fetching smaller chunks from the servers is expected to take shorter. However, a file download requires fetching all its k chunks and higher k quickly increases the chances of a slow download due to the service time variability [1]. 2) *Redundancy-$(n - k)$*: Files are downloaded by requesting both the k original and the $n - k$ coded chunks simultaneously, so that the fastest k downloads is sufficient for file recovery. 3) *Load balancing-N/n*: Request arrivals are distributed across N/n sub-systems, so expanding the size n of the sub-systems causes higher load per server, which is known to cause increasingly larger queueing delays.

In the following, we study the combined effect of all the above factors on the average download time. Response time analysis of (n, k) system is formidable for general k. Two main reasons are that system state space explodes exponentially in k and multiple requests can be in service simultaneously. An upper bound on the response time is derived in [4] by treating

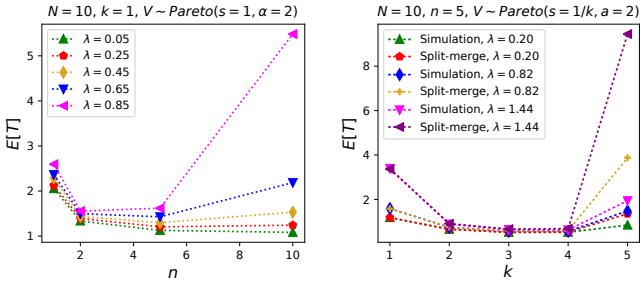

Fig. 1. Left: Average download time vs. level of replication n in $(N = 10, n, k = 1)$ system under different arrival rates λ. Right: Average download time vs. level of parallelism k in $(N = 10, n = 5, k)$ system. Values from the simulations and the split-merge approximation are plotted for varying λ.

the system as if it implements *split-merge* service model. Under split-merge, each arriving request is again copied into all n servers, however servers that go idle cannot continue with serving the request copies in their queues. In other words, only the copies of the request at the head of the line in the system are allowed to be in service simultaneously. Then, service time of each request becomes i.i.d. and the system works as an $M/G/1$ queue with request service times distributed as the kth order statistics of n i.i.d. server service time V's; $V_{n:k}$. When $k = 1$, remaining $n-1$ copies of a request get canceled as soon as the fastest copy completes, hence the $(n, 1)$ system and the split-merge model become the same.

When $k = 1$ (no parallelism), download time for an arbitrary request in $(N, n, 1)$ system is equivalent to the response time of an $(n, 1)$ replicated storage under Poisson arrivals at rate $\lambda' = \lambda n/N$. Each $(n, 1)$ sub-system is equivalent to an $M/G/1$ queue with request service times distributed as $V_{n:1}$. Then, the PK formula gives the average download time as

$$E[T] = E[V_{n:1}] + \frac{\lambda' E[V_{n:1}^2]}{2(1 - \lambda'/E[V_{n:1}])}$$

Fig. 1 (Left) plots the average download time for increasing values of n and under different arrival regimes λ. Recall that higher n increases the level of download redundancy but also incurs higher load on the servers. As illustrated in Fig. 1, at higher arrival rate regimes, increasing n quickly overburdens the system with redundant requests and causes further slowdown in download times. Under all arrival regimes, relative reduction in download time by redundancy is highest at the first incremental step, namely going from $n = 1$ (no redundancy) to $n = 2$ (one replica for each file). Underlying reason is the order statistics; adding more samples into the population gives diminishing return in reducing the value of the minimum. Effect of the greater server load due to redundant requests cannot be compensated by the reduction in service time variability at higher values of n.

When $k > 1$, files are split into k chunks of equal size. It is not clear how service times scale with parallelism k in practice. We adopt a model in which service times are linearly related to the data size, e.g., if service time for the whole file is kV, then with parallelism k, service time for one of its chunks is V. Approximating each (n, k) sub-system with split-merge service model, the PK formula gives the following upper bound on the average download time for an arbitrary

request arriving to the (N, n, k) system.

$$E[T] = E[V_{n:k}] + \frac{\lambda' E[V_{n:k}^2]}{2(1 - \lambda'/E[V_{n:k}])}$$

Fig. 1 (Right) plots the simulated average download time and its split-merge upper bound for fixed n and varying k. Although split-merge becomes a looser approximation for larger k and λ, it is able to capture the change in download time with respect to k. At all arrival rates, reduction in average download time with higher parallelism is most significant as we go from $k = 1$ (replicated storage) to $k = 2$ ($(n, 2)$ coded storage), which is especially the case for small files. Increasing k beyond a value hurts performance since the level of redundancy $n-k$ decreases and the slowdown due to service time variability starts to exceed the gain of parallelism.

Low parallelism is also preferred in practice to reduce the operational complexity and overhead e.g., fewer chunks help to reduce the number of control actions and the contacted servers while executing chunk query, chunk read/write etc [14]. Another practical reason for low parallelism is that new storage codes are desired to have low locality; the number of nodes accessed while recovering from a single node failure [10]. Both parallelism and locality of (n, k) MDS codes are equal to k, which is desired to be low in practice. Note that, lower k for fixed n reduces the storage efficiency of MDS codes, which is another important (may be the most important of all in some cases) concern while deploying storage codes.

III. Approximating download time

Fork-join queues and their generalization (n, k) systems are very challenging to analyze because their state space explodes exponentially with k. Also multiple download requests can be in service simultaneously and different copies of a request may start service at different times. Next, we state two important observations that give insight for approximating the system.

Observation 1 (Request departure order): Requests depart an (n, k) system in the order they arrive.

At all times, at least $n - k + 1$ of the servers serve the copies of the same request in the system, which we refer to as the *HoL (head of the line) request*. Note that HoL request is the oldest request in the system. This is because, at most $k-1$ copies of a request can finish before it departs. This means, at most $k-1$ copies of a waiting request can start service before it becomes the HoL request, and thus, no waiting request can depart before the HoL request. The next observation follows naturally.

Observation 2 (Request service start): We define the service start time of a request as the epoch at which each of its copies either has departed or is in service, i.e., as soon as it becomes the HoL request. Some copies of a request may start service or even depart before the request moves to HoL. At least $n - k + 1$ and at most all n copies of a request can be simultaneously in service when it is at the HoL.

When server service times are exponential, Obv. 2 can be updated by using the memoryless property as follows.

Claim 1 (Request service time distributions): When server service time V's are i.i.d. exponential, service time of an arbitrary request in (n, k) system is distributed as $V_{n-j:k-j}$ where j can be any integer in $[0, k-1]$.

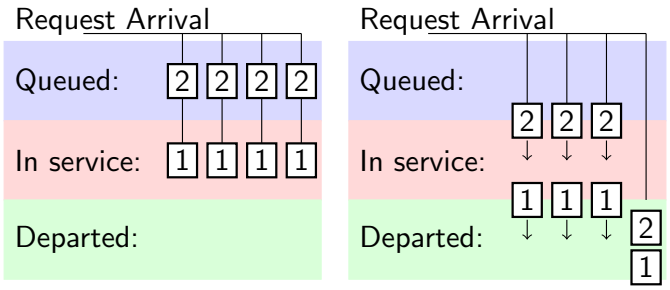

Fig. 2. Two possible request service times in $(4, 2)$ system. (a) All copies of request 1 are in service and any two is sufficient for its completion. (b) One copy of request 2 departs earlier, the remaining three copies move into service as soon as request 1 departs, and any two of its remaining copies is sufficient for completing request 2.

Note that this observation does not hold for general V since some of the copies may start service earlier and remain in service until the request moves to HoL. In the absence of memoryless property, residual lifetime of this early starting copies would be different from those that move in service together with all the rest once the request moves to HoL.

Service time distribution of a request in (n, k) system is completely determined by the system state at the time epoch at which the request moves to HoL. Since the system state is dependent over the course of multiple request service starts, request service times are not independent. The following observation is good news for treating the request service times "approximately" independent.

Observation 3 (Chain of dependence breaks): When server service time V's are exponential, time epochs at which a request starts service and has all its n copies in service, break the dependence between the requests that start service before and those that start service after. For general V, these time epochs only loosen the dependence.

This observation is essential for the approximate method that is presented in the remaining of this Section.

A. Analysis of $(n, 2)$ system

We firstly assume that server service times are exponentially distributed; $V \sim \text{Exp}(\mu)$. Then, only two request service time distributions are possible in $(n, 2)$ system: 1) *Complete start*: all copies of a request start service simultaneously, so its service time is $V_{n:2}$, 2) *Partial start*: one of its copies departs before the request moves to HoL, so its service time is $V_{n-1:1}$. Fig. 2 illustrates these two possibilities for the $(4, 2)$ system.

Obv. 3 can be expanded for $(n, 2)$ system. There can be at most one *leading server* with a shorter queue than the rest of the other servers. This is because given a leading server, as soon as a request copy is completed at one of the other servers, the request at the HoL departs and the difference between the queue lengths of the leading and non-leading servers diminish by one. In fact, the leading server has to compete to keep leading and need to execute the request copies faster than the fastest one of the $n-1$ non-leading servers. This can happen

with low probability since the service time at the leading server is stochastically greater than the minimum of $n-1$ samples from the non-leading servers, which suggests that the queue lengths will frequently level up. Thereby, the time epochs that decouple request service times will be frequent (see Obv. 3). Note that, this observation does not generalize for general k. In general, there can be at most $k-1$ leading servers and it is hard to derive a stochastic ordering between the service completion at the leading and the non-leading servers.

Proposition 1 (Approximate method): When server service times V are exponentially distributed, $(n,2)$ system can be approximated as an $M/G/1$ queue with request service time B distributed as

$$\begin{aligned} Pr\{B>v\} = Pr\{V_{n:2}>v\}Pr\{S=c\} \\ + Pr\{V_{n-1:1}>v\}Pr\{S=p\}. \end{aligned} \quad (1)$$

where S denotes the complete (c) or partial (p) request service starts.

Observations we have made so far allow us to develop an approximate method for the analysis of $(n,2)$ system. Requests depart in the order they arrive (Obv. 1) hence the system acts as a first-in first-out queue with possible request service times given in Claim 1. Although request service times are not independent, they exhibit loose coupling (Obv. 3). Relying on this and assuming i.i.d. request service times gives us an $M/G/1$ approximation to the actual system. Then, one needs to know the distribution B of request service times. We know there are two possible distributions: $V_{n:2}$ when service start type S is complete and $V_{n-1:1}$ when S is partial. In the following, we discuss a method to estimate the distribution of S.

B. Markov process for $(n,2)$

Think of a join queue at the tail of the system where the departed request copies wait for their siblings to finish service (see the departed region in Fig. 2). We define the state of the system at time t by $(\mathcal{N}(t),\mathcal{D}(t))$ where $\mathcal{N}(t)$ is the length of the longest queue in the system and $\mathcal{D}(t)$ is the the difference between the longest and the shortest. Only the request copies that departed from the same server can be in the join queue simultaneously. For instance in Fig. 2, system state is $(\mathcal{N}=2,\mathcal{D}=0)$ on the left and $(\mathcal{N}=1,\mathcal{D}=1)$ on the right side. Markov process for the system state is illustrated in Fig. 3 (Left), which is infinite in two dimensions so its analysis is hard.

C. High-traffic approximation

Suppose that the arrival rate λ is very close to its critical value for stability and the system never goes idle. This is a rather crude assumption and holds only when the system is unstable. Under this assumption, state of the join queue $\mathcal{D}(t)$ itself is a simple birth-death chain as shown in Fig. 3 (Right). We refer to this set of working conditions as *high-traffic approximation*. Service completions at the leading server correspond to the state transitions towards right, while the completions at the non-leading servers represent request departures and correspond to the transitions towards left. Service cancellations at the servers due to request departures are

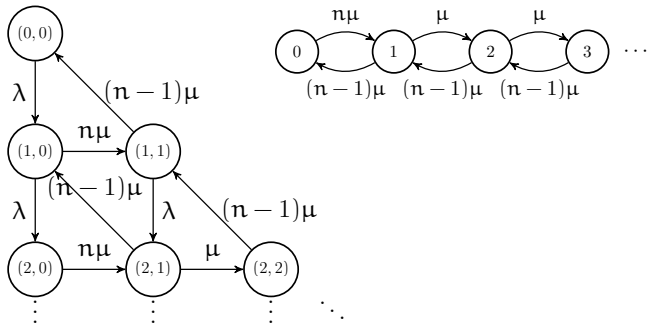

Fig. 3. Markov process for the $(n,2)$ system (Left) and its high-traffic approximation (Right) assuming rate-μ exponential server service times.

not registered as a service completion. Steady state balance equations for the birth-death process are

$$n\mu p_0 = (n-1)\mu p_1, \quad \mu p_i = (n-1)\mu p_{i+1}, \quad i>0.$$

where $\lim_{t\to\infty} Pr\{\mathcal{D}(t)=i\} = p_i$. Solving the balance equations gives

$$p_0 = [1+n/(n-2)]^{-1} \text{ and } p_i = n(n-1)^{-i}p_0 \text{ for } i\geq 1. \quad (2)$$

We will use the Markov Chain embedded in the approximate birth-death process to estimate the distribution of request service start type S as required in (1) for the $M/G/1$ approximation. Since the total transition rate ($n\mu$) out of each state is the same, steady state probabilities π_i's for the embedded chain are equal to p_i's.

Under high traffic assumption, system is always busy and a complete request service start occurs for every transition into state-0, while a partial request service start occurs for every transition into any other state. Define f_d as the fraction of state transitions that represent request departures and $f_{\to c}$ as the fraction of state transitions that define a complete request service start. Then we find them as

$$\begin{aligned} f_d &= \sum_{i=1}^{\infty} \frac{n-1}{n}\pi_i = \frac{n-1}{n}(1-\pi_0), \\ f_{\to c} &= \frac{n-1}{n}\pi_1 = \frac{n-1}{n}\frac{n}{n-1}\pi_0 = \pi_0. \end{aligned}$$

Then the limiting fraction of the requests that make partial (f_p) or complete (f_c) start are found as

$$\begin{aligned} f_c &= \frac{f_{\to c}}{f_d} = \frac{n\pi_0}{(n-1)(1-\pi_0)} = 1-1/(n-1), \\ f_p &= 1-f_c = 1/(n-1). \end{aligned}$$

It is worth to notice that, the fraction of the requests that make complete start increases with n, hence the system behaves more like the split-merge model when n is large. This is natural since the leading servers have to compete with more servers when n is larger, and the probability and consequently the fraction of partial request service starts decline.

We use the limiting fractions f_c and f_p to estimate the probabilities $Pr\{S=c\}$ and $Pr\{S=p\}$. Substituting them in (1) gives the service time distribution required for the $M/G/1$ approximation. Fig. 4 gives a comparison of the

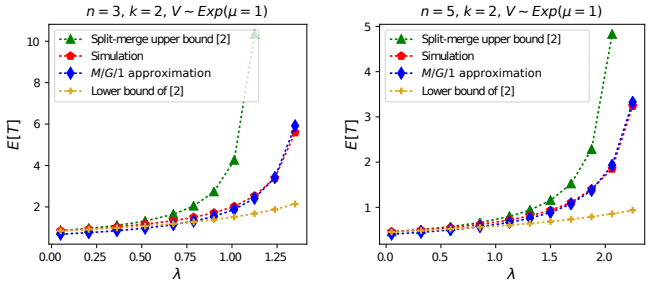

Fig. 4. Comparison of the upper and the lower bounds of [4], and the average download time from the simulations and the $M/G/1$ approximation for $(n, 2)$ system with $n = 3, 5$ and exponential service times.

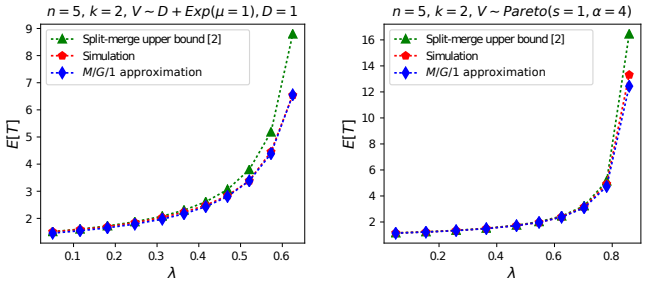

Fig. 5. Average download time in $(n, 2)$ system for non-exponential service times. Simulations, split-merge upper bound given in [4] and the $M/G/1$ approximation are compared.

simulated average download time with its approximated values. Approximation performs overall better than the split-merge upper bound and the lower-bound derived in [4], however it performs slightly worse for low arrival rates. Split-merge is a very good approximation for low arrival rates since the system cannot feed the idle servers to go ahead with the copies of the waiting requests, hence the system cannot deviate much from the split-merge behavior under low arrival rate. As expected, high traffic approximation is more accurate for increasing values of λ.

D. General service times

Claim 1 is not valid when server service time V's are not exponential. The reason is that a copy of a request may start service early and remain in service until the request moves to HoL. Residual service time of these early starters is not the same as the service time of the fresh starters, unless V is memoryless. However, one can modify the system and restart the early starters once a request moves to HoL. This modification increases or reduces download time depending on whether V has increasing or decreasing hazard rate.

Another caveat is estimating the distribution of request service start type S. When V is not exponential, state space of the system is intractable, even under high traffic assumption. The estimates f_c and f_p that were previously found for exponential V are simple functions of n. They are the best estimates we have, and we use them for approximating the distribution of S for non-exponential V. Fig. 5 plots the simulated and approximated values of the average download time in $(n, 2)$ system when $V \sim$ S-Exp or $V \sim$ Pareto. While the approximation is doing better than the split-merge upper bound, the difference is not large because the performance of the actual system is close to that of the split-merge service model. The reason is that the non-zero minimum value (D for S-Exp and s for Pareto) of the service times does not allow the leading server to go ahead far in serving the waiting request copies and the system cannot perform much better than simply blocking the idle servers as in split-merge. Note that the lower bound given in [4] is not available for non-exponential V's.

ACKNOWLEDGMENTS

This research is supported by the National Science Foundation under Grants No. CIF-1717314.

REFERENCES

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

[2] K. Gardner, S. Zbarsky, S. Doroudi, M. Harchol-Balter, and E. Hyytia, "Reducing latency via redundant requests: Exact analysis," *ACM SIGMETRICS Perform. Evaluation Review*, vol. 43, pp. 347–360, 2015.

[3] K. Gardner, M. Harchol-Balter, and A. Scheller-Wolf, "A better model for job redundancy: Decoupling server slowdown and job size," in *Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS), 2016 IEEE 24th International Symposium on*. IEEE, 2016, pp. 1–10.

[4] G. Joshi, Y. Liu, and E. Soljanin, "Coding for fast content download," in *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*. IEEE, 2012, pp. 326–333.

[5] R. Nelson and A. N. Tantawi, "Approximate analysis of fork/join synchronization in parallel queues," *IEEE transactions on computers*, vol. 37, no. 6, pp. 739–743, 1988.

[6] G. Joshi, E. Soljanin, and G. Wornell, "Efficient redundancy techniques for latency reduction in cloud systems," *ACM Transactions on Modeling and Performance Evaluation of Computing Systems (TOMPECS)*, vol. 2, no. 2, p. 12, 2017.

[7] L. Huang, S. Pawar, H. Zhang, and K. Ramchandran, "Codes can reduce queueing delay in data centers," in *Proceed. 2012 IEEE International Symposium on Information Theory (ISIT'12)*, pp. 2766–2770.

[8] G. Liang and U. C. Kozat, "Fast cloud: Pushing the envelope on delay performance of cloud storage with coding," *IEEE/ACM Transactions on Networking*, vol. 22, no. 6, pp. 2012–2025, 2014.

[9] N. B. Shah, K. Lee, and K. Ramchandran, "The mds queue: Analysing the latency performance of erasure codes," in *Proceed. 2014 IEEE Internat. Symposium on Inform. Theory (ISIT'14)*, pp. 861–865.

[10] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols," *IEEE Transactions on Information Theory*, vol. 58, no. 11, pp. 6925–6934, 2012.

[11] S. Kadhe, E. Soljanin, and A. Sprintson, "Analyzing the download time of availability codes," in *Information Theory (ISIT), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 1467–1471.

[12] ——, "When do the availability codes make the stored data more available?" in *Communication, Control, and Computing (Allerton), 2015 53rd Annual Allerton Conference on*. IEEE, 2015, pp. 956–963.

[13] M. F. Aktas, E. Najm, and E. Soljanin, "Simplex queues for hot-data download," in *Proceedings of the 2017 ACM SIGMETRICS/International Conference on Measurement and Modeling of Computer Systems*. ACM, 2017, pp. 35–36.

[14] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The google file system," in *ACM SIGOPS operating systems review*, vol. 37, no. 5. ACM, 2003.

[15] D. Borthakur, "The hadoop distributed file system: Architecture and design," *Hadoop Project Website*, vol. 11, no. 2007, p. 21, 2007.

[16] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz, and M. A. Kozuch, "Towards understanding heterogeneous clouds at scale: Google trace analysis," *Intel Science and Technology Center for Cloud Computing, Tech. Rep*, p. 84, 2012.