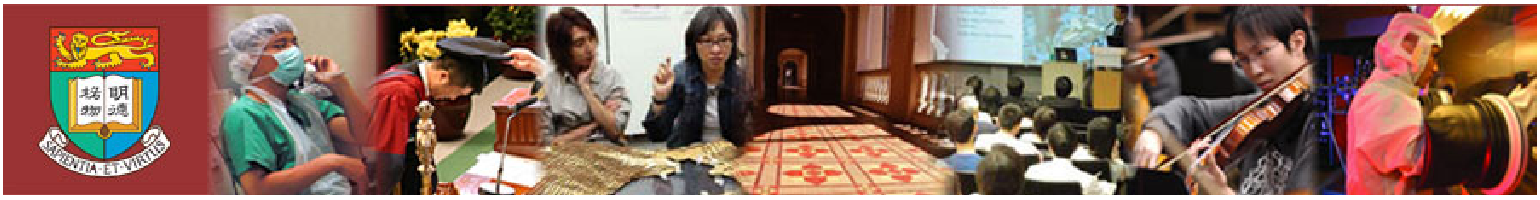


Latency Performance Model of Direct and K-access Reads in Distributed Storage Systems

Qiqi Shuai
Department of Electrical and Electronic Engineering
The University of Hong Kong
Hong Kong, China
Email: qqshuai@connect.hku.hk

Victor O.K. Li, *Fellow, IEEE*
Department of Electrical and Electronic Engineering
The University of Hong Kong
Hong Kong, China
Email: vli@eee.hku.hk

***Abstract*—Access latency is a crucial performance metric in distributed storage systems as it greatly impacts user experience, especially for hot data visitors. Existing papers argue that coding can reduce access latency compared with the replication method, and many dispatch schemes, such as those implementing redundant requests, dynamically changing code rates, and so on, are proposed and proved to work well in improving latency performance under certain conditions. However, some practical issues, such as direct reads, and the additional cost of redundant requests, are ignored and a general and practical model of latency performance is still lacking.**

Considering a more comprehensive set of practical issues in distributed storage systems, we propose a performance model that can be easily used to compare latency performance of different codes and replication methods under different conditions. We also use our performance model to evaluate many schemes and show their different impacts on the latency performance of different types of reads. To the best of our knowledge, we are the first to study the latency performance of direct reads under different codes or schemes, and the first to propose a model of latency performance including both direct and k-access reads.

***Keywords*-MDS code; access latency; replication; direct reads; k-access reads; redundant requests;**

I. Introduction

In massive distributed storage systems, failure is the norm rather than the exception [1]. To tolerate frequent failures and provide sufficient reliability and availability of data, we need to increase storage redundancy. Replication and erasure coding are the usual methods used.

A single codeword of an erasure code has n fragments, k of which are original data fragments, and the other $n-k$ are parity fragments. If a code satisfies the maximum-distance-separable (MDS) property, any k out of the n fragments are sufficient to reconstruct all data in the k original data fragments.

Access latency is a key performance metric for distributed storage systems and has great impact on user experience, especially for data retrieval applications [2]. For instance, 500 msec extra delay in service will lead to a 1.2% user loss for Google and Amazon [3]. However, most papers focus on other performance metrics such as storage overhead, repair cost and so on. Only recently do some papers focus on access latency in storage systems and suggest that coding can reduce access latency.

Huang $et\ al.$ [4] analytically compared the latency performance of replication and coding when all data are divided into $k = 2$ parts and argued that coding can reduce queueing delay in data centres. Following that work, based on the MDS code, with queueing theory, Shah $et\ al.$ [5] proposed different scheduling policies to demonstrate the effectiveness of coding on reducing access latency. Our previous work [6] proposed DRALB scheme to reduce access latency of hot data in systematic coded storage systems. Vulimiri $et\ al.$ [7] argued that redundant requests in the context of the wide-area Internet can help reduce latency. A theoretical analysis [8] discussed the conditions under which redundant requests can help reduce access latency. Based on fork-join queues for parallel processing, Joshi $et\ al.$ [9] generalized the (n, n) fork-join system and found bounds on its mean response time. In [10], we proposed the compound read method, characterized its mean download delay in low arrival rate scenario and derived upper and lower bounds on its mean download delay in high arrival rate scenario. Liang $et\ al.$ [11] presented a strategy TOFEC, using erasure coding, parallel connections and limited chunking together, to improve the delay performance. McCullough $et\ al.$ [12] developed Stout to dynamically increase or decrease batching size to improve access latency. We designed new codes HTSC and FH_HTSC [13] to reduce access latency while maintaining favorable performance in other metrics. Liang $et\ al.$ [14] proposed dynamically changing code rate to reduce access latency.

Although many different dispatch schemes have been proposed recently and they have somehow been proved to reduce access latency in distributed storage systems under certain conditions, a general and practical performance model of access latency is still lacking. This makes it exceedingly arduous to compare the effects of different schemes and to convince others that those schemes can really work under different conditions. Besides, some practical issues are also ignored in much of existing research.

First, based on MDS-coded system, other than some work that mentioned direct reads [15], [16], almost all previous studies of latency performance assume that each request to the storage system needs to access at least k storage nodes (k-access reads). However, in practice, the code deployed is usually a systematic code, which means that one copy of the data exists in uncoded form [17]. Besides, in many storage systems such as Windows Azure Storage (WAS), only when a file reaches a certain size (e.g., 3GB), will it be a candidate for erasure coding [15]. That is, one file in a codeword is usually extremely large and incoming requests may only desire part of the file, say, the systematic part that is stored in one of the storage nodes (we call those requests direct read requests). Actually, both k-access and direct read requests commonly exist in MDS-coded systems but they require different sizes of files, thus triggering different access latencies. Essentially, most previous research assumes homogeneous read requests but in practice, requests may desire files of different sizes.

Second, while degraded reads [18], [19] are common in distributed storage systems, they are ignored by almost all previous work on access latency. Degraded reads occur when one storage node is too busy serving other requests and becomes temporarily unavailable to a new direct read request, and we need to reconstruct the required data from other nodes. Obviously, degraded reads can also influence user experience and must be included in the access latency analysis. Although Shah $et\ al.$ [5] pointed out the existence of degraded reads and analysed their access latencies in product-matrix (PM) coded system [20], they did not consider the relationship of degraded reads with other requests.

Third, some performance models were designed just to evaluate specific schemes, and it is not applicable for evaluating other schemes. As an example, using a novel fork-join queueing framework to model multiple users requesting the content simultaneously, Joshi $et\ al.$ found bounds of mean response time in [9]. However, the model in [9] can only be used in MDS-coded storage systems with Redundant Request Scheme (RRS). Moreover, that paper only considers the k-access read requests while direct reads and the additional cost of redundant requests are ignored.

Our Contributions. In this paper, we propose a general and practical model to analyze latency performance in distributed storage systems. To the best of our knowledge, our model is the first to consider all types of read requests together, and can be used for different types of codes and different dispatch schemes. With our model, we compare the latency performance of coding and replication under different conditions, and we also show more practical latency performance of different schemes. We are the first to study the latency performance of direct read requests under different codes or schemes, and the first to propose a performance model of latency including both direct and k-access reads and considering many more practical issues such as the additional cost of redundant requests.

The remainder of this paper is organised as follows. In Section II we describe the system model in detail. In Section III we show how to apply different schemes and codes in our model and further discuss RRS. Section IV displays simulation results and shows the effect of different schemes and codes on reducing access latency when considering many more practical issues. Finally, in Section V, we conclude and discuss some future work.

II. System Model

A. Basic Architecture

A distributed storage system is mainly composed of one dispatcher (possibly more than one in practice) and massive distributed storage nodes. As illustrated in Fig. 1, a quintessential work flow is given as follows: different user requests (read and write) arrive at a dispatcher, which distributes the requests as different tasks according to users' requirements to different storage nodes via the inter or intra data centre network.

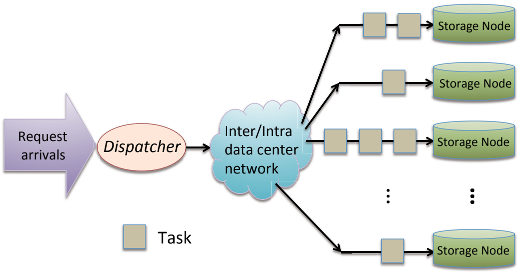


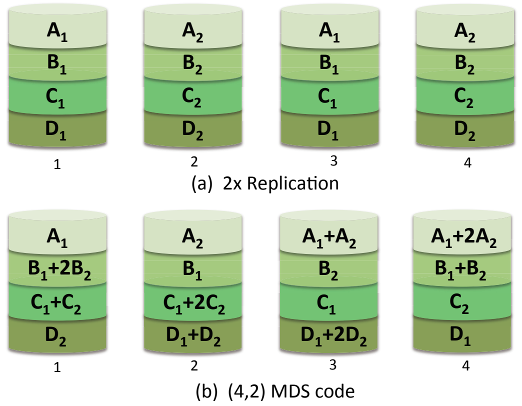
Figure 1. System model.

In practice, the dispatcher may execute different schemes to achieve load balancing and improve the reliability and availability of the distributed storage systems.

B. Storage Nodes

In practical distributed storage systems, each storage node contains many fragments and with erasure coding, one node may store both data and parity fragments. As illustrated in Fig. 2, each file A, B, C and D is divided into two fragments and each node can store four fragments. For the 2x replication method, there are two copies of all fragments. For (4,2) MDS code, each file is encoded into 4 fragments and each node stores both data and parity fragments. Fragments from the same file or codeword will never be stored in a single node so as to reduce the probability of their simultaneous failure, since failure usually happens in the whole node.

In this paper, we propose a performance model of access latency based on (n, k) MDS code. We focus on the latency performance of n storage nodes which belong to the same codeword, and we call it an n-node structure. Since a storage system is composed of many n-node structures, this model



(a) 2x Replication

(b) (4,2) MDS code

Figure 2. Four storage nodes with 2x replication and (4,2) MDS code.

can reflect the access latency of the whole storage system. We also assume all the codes in this paper is systematic, which is in line with almost all practical codes.

C. Characteristics of Different Requests

In distributed storage systems, there are usually two kinds of requests, namely, read and write. Here we consider a typical append-only distributed storage system such as HDFS [21] or WAS [15]. In such systems, when we save something, the dispatcher can quickly distribute it to some applicable idle storage nodes. Users often do not care much about the delay of their write requests as long as they can be done within a reasonable period. But things are totally different for read requests. The access latency of read requests can greatly impact user experience. For example, Google found that users performed up to 0.74% fewer searches after a 400 millisecond additional delay has been implemented for 4 to 6 weeks [22]. Besides, since there are only limited nodes in the system storing the data desired by read requests, higher frequency of read requests will inevitably increase the access latency. Also, in an MDS-coded system, since we will break each file into k parts, encoded into n coded elements and stored in n storage nodes, for each write request, we need to write into at least k nodes and then reply that we have done it [14]. That is, the analysis of write request is similar to that of k-access read request. Accordingly, in this paper, we focus on the access latency of read requests (Note that our results can also be applied to write requests).

Read requests can be divided into three types of tasks for a storage node: k-access, direct and degraded read tasks, as illustrated in Fig. 3. A k-access read desires one complete file, such as the file A, B, C or D in Fig. 2. The request has to access at least k fragments in an (n, k) MDS-coded system. As discussed in [5], incoming requests may also require only part of the file, say, the part that is stored in one of the storage nodes, such as A_1 or B_1 in Fig. 2. Since the code is systematic, data retrieval can be done by reading it directly from one fragment. We call this a direct read. But when a node is too busy, we can serve part of the direct read requests via degraded reads. In an MDS-coded system, a degraded read task may be performed by obtaining the entire file from the data stored in any k out of the $n-1$ remaining fragments and then extracting the required part. When we build the performance model, we need to carefully consider the characteristics of different read requests simultaneously.

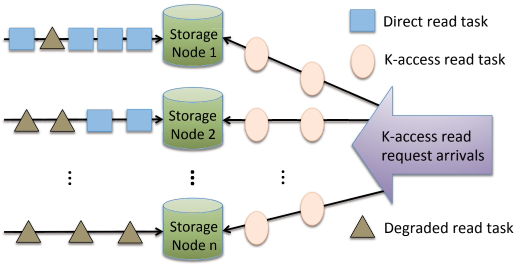


Figure 3. Different types of read tasks to storage nodes.

D. Arrival Rates of General Read Tasks

In this paper, we focus on the case that we can operate different fragments in a node in parallel, that is, different tasks actually queue in front of different fragments in each node. Then we just need to examine the performance of one fragment in each storage node to get the average latency performance in the n-node structure. Without loss of generality, we focus on the requests to the first fragment in each node, such as A illustrated in Fig. 2. Suppose the direct read arrival rate of the first fragment in the jth storage node is λ_{aj}, where $j = 1, 2, ..., n$. Suppose x_j is the fraction of direct reads and $1-x_j$ is the fraction of degraded reads by the other $n-1$ fragments in the same codeword. Accordingly, the actual direct read arrival rate of this fragment is $x_j e_j \lambda_{aj}$ where $e_j = 1$ when the first fragment in the jth node is a data fragment and $e_j = 0$ when the first fragment in the jth node is a parity one, since there is no direct read to parity fragments in practice. The degraded read arrival rate to the first fragment in the jth node from the other $n-1$ nodes is $(1-x_1)p_a e_1 \lambda_{a1} + ... + (1-x_{j-1})p_a e_{j-1} \lambda_{a(j-1)} + (1-x_{j+1})p_a e_{j+1} \lambda_{a(j+1)} + ... + (1-x_n)p_a e_n \lambda_{an}$, where $p_a = \frac{k}{n-1}$ in (n, k) MDS-coded systems and p_a can take different values according to different types of codes or specific dispatch schemes. Suppose the k-access read arrival rate to file A is λ_A, then the k-access read arrival rate for the first fragment in each node is $p_A \lambda_A$, where $p_A = \frac{k}{n}$ on the average in (n, k) MDS-coded systems. p_A can also take other values according to the code types and dispatch schemes. Considering all the three types of read together, we can get the general read arrival rate of this fragment in the jth storage node as $\lambda_{aj}{}' = x_j e_j \lambda_{aj} + (\sum\limits_{m=1}^{n} (1-x_m) p_a e_m \lambda_{am} - (1-x_j) p_a e_j \lambda_{aj}) + p_A \lambda_A$ and the average general read arrival rate for the first fragment in all the n nodes as $\overline{\lambda_a'} = \frac{1}{n}\sum_{j=1}^{n}[x_j e_j \lambda_{aj} + (n-1)(1-x_j)p_a e_j \lambda_{aj}] + p_A\lambda_A$.

The result of the general read arrival rate is not influenced by the distributions of request arrivals and services, and consequently, we can apply it under different conditions.

E. Latency Analysis

The Random Dispatch Scheme (RDS) is the default scheme in our model, and its key idea is that for each k-access read request, the dispatcher randomly distributes it to any k out of n nodes and for a degraded read, the dispatcher randomly distributes it to any k out of the $n-1$ remaining nodes. Obviously, this scheme is very simple and does not need other information or resources, thus making it the default scheme in many practical distributed storage systems.

From the results of real traces over Amazon S3 in [3], [14], it is observed that there is negligible correlation between the service times of different requests. Accordingly, we will treat request service times as independent and identically distributed ($i.i.d.$) variables.

In our model, we dispatch requests to different storage nodes and can easily calculate different task arrival rates for each node. Consequently, for each storage node in an n-node structure, we can construct a queueing system with one single server to study its access latency separately. Then we can obtain the latency performance of the whole n-node structure.

Theorem 1: In a systematic (n, k) MDS-coded storage system with RDS, the average latency of general read tasks of each node in an n-node structure is W_{ai}, $i = 1, 2, ..., n$. Suppose we have ascendingly sorted W_{ai}, then the average access latency of k-access read requests is $\frac{1}{n-k+1}\sum_{i=k}^{n} W_{ai}$.

Proof: In a (n, k) MDS-coded storage system, some fraction of direct reads can be transferred to degraded reads. Even though there is no direct read request to the parity fragments, they will indirectly serve direct reads by helping finish degraded read tasks.

However, since each k-access read request will be dispatched to any k of the n nodes, its access latency is determined by the largest access latency from the k nodes. Since we have sorted W_{ai} in ascending order, each latency W_{ai}, $i = k, k+1, ...n$ can be the longest access latency from k nodes with equal probability, so the average access latency of k-access read requests is $\frac{1}{n-k+1}\sum_{i=k}^{n} W_{ai}$. The count method here is in line with the model in [14]. (Note that we suppose random dispatch here, for other schemes such as RRS, each request will be dispatched to all the n nodes.) ■

Theorem 2: In a systematic (n, k) MDS-coded storage system with RDS, for any data fragment in the jth node, x_j is the fraction of direct reads and $1 - x_j$ is the fraction of degraded reads by the other $n-1$ fragments in the same codeword. Suppose the average latency of the data fragment is W_{aj}, $j = 1, 2, ..., k$ and the average latency of the other $n-1$ fragments in a codeword is W_{ai}, $i = 1, 2, ..., n-1$ and we have ascendingly sorted W_{ai}, then the average access latency of direct read requests to the data fragment in the jth node is $x_j W_{aj} + (1-x_j)\frac{1}{n-k}\sum_{i=k}^{n-1} W_{ai}$, where $j = 1, 2, ..., k$.

Proof: In the background of Theorem 2, the fraction x_j of direct reads directly go to the data fragment in the jth node, and obviously, the average access latency is W_{aj}. While the fraction $1-x_j$ of direct reads of the data fragment in the jth node are transferred as degraded reads to the other $n-1$ fragments in the same codeword except for the data fragment in the jth node. In RSD, the degraded reads randomly dispatch each request as degraded read task to any k out of the $n-1$ remaining fragments and consequently, similar to the proof of Theorem 1, the average latency of degraded reads is determined by the largest access latency from the remaining $n-1$ fragments. Since we have sorted W_{ai}, $i = 1, 2, ..., n-1$ of the remaining $n-1$ fragments in ascending order, each latency W_{ai}, $i = k, k+1, ...n-1$ can be the longest access latency from k nodes with equal probability, so the average access latency of degraded reads is $\frac{1}{n-k}\sum_{i=k}^{n-1} W_{ai}$. Accordingly, with the fractions of direct read tasks and degraded read tasks, we can get the average access latency of direct read requests to the data fragment in the jth node is $x_j W_{aj} + (1-x_j)\frac{1}{n-k}\sum_{i=k}^{n-1} W_{ai}$, where $j = 1, 2, ..., k$. ■

III. Model Applications

(1) Applications to Different Codes and Schemes

The key idea of our performance model is to use different values of p_a and p_A to reflect almost all the key features of different types of codes and different dispatch schemes. Some specific applications of our performance model to various codes and schemes are summarized in Table I.

Even though additional requests can be cancelled after the first k tasks have been finished, RRS will inevitably increase the actual task arrivals to each node. $\xi \geq 1$ is a parameter to properly increase the value of p_a and p_A to reflect the additional cost of redundant requests, and it can be adjusted according to the practice. For example, $\xi = 1$ means no additional cost and $\xi = 1.1$ can represent 10% additional cost since it increases the workload by 10% for each node. Previous work [23] assumes the additional cost of redundant request will decrease the service rate of each storage node, while we suppose that additional cost will increase the actual request arrival rate (system load), and we will show that our model captures more of the reality in the evaluation section.

(2) Further Discussion of RRS

Table I
SUMMARY OF APPLICATIONS TO DIFFERENT CODES AND SCHEMES

Code Type	Scheme	p_a	p_A
$(c, 1)$Replication	Random dispatch	$\frac{1}{c-1}$	$\frac{1}{c}$
$(c, 1)$Replication	Redundant requests	$\xi\frac{1}{c-1}$	$\xi\frac{1}{c}$
$(c, 1)$Replication	Changing code rate	$\frac{1}{C-1}$	$\frac{1}{C}$
(n, k)MDS Code	Random dispatch	$\frac{k}{n-1}$	$\frac{k}{n}$
(n, k)MDS Code	Redundant requests	$\xi\frac{k}{n-1}$	$\xi\frac{k}{n}$
(n, k)MDS Code	Changing code rate	$\frac{k}{N-1}$	$\frac{k}{N}$
Note: c is the number of copies with Replication, ξ is a parameter to reflect the additional cost of redundant requests.			

Recently, a lot of work demonstrates that we can reduce access latency by sending redundant requests in distributed storage systems. Some of the work [5], [8], [9] are based on theoretical analysis, while some [11], [14] are also based on trace-driven simulations. However, almost all the previous work supposes that all requests are k-access requests and ignores direct reads. However, with our model, similar to RDS, we can easily obtain the average access latency of both k-access and direct read requests with RRS.

Corollary 1: With RRS, the average access latency of k-access read requests is W_{ak} and the average access latency of direct read requests to the data fragment in the jth node is $x_j W_{aj} + (1 - x_j)W_{ak}$, where $j = 1, 2, ..., k$.

It is straightforward that with RRS, the access latency of k-access read requests depends on the kth smallest value among W_{ai}, $i = 1, 2, ..., n$, that is W_{ak} and similarly, the access latency of degraded reads relies on kth smallest value among W_{ai} of the other $n - 1$ fragments. Then based on the results of RDS in Theorems 1 and 2, we can easily get the results in Corollary 1.

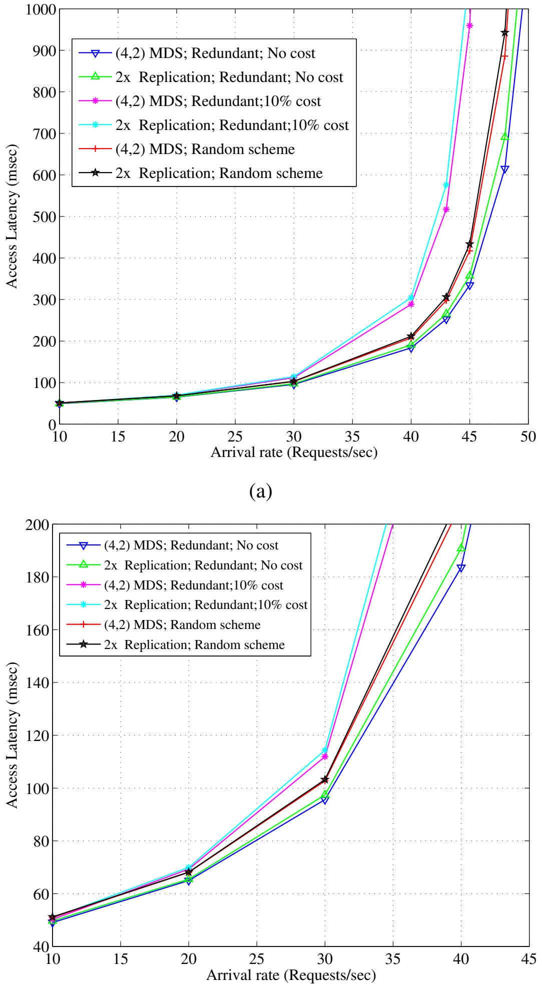
From Theorem 1 and Corollary 1, we can see that the average latency of k-access read request depends on the mean value of the largest $n - k + 1$ values among W_{ai}, $i = 1, 2, ..., n$ in RDS while in RRS, it relies on the kth smallest value among W_{ai}, $i = 1, 2, ..., n$, that is W_{ak}. Clearly, RRS is superior to RDS. Similarly, from Theorem 2 and Corollary 1, we can also easily see the advantage of RRS over RDS in terms of the latency performance of direct reads. The degree of superiority depends on the value of the direct read fraction x_j, with the lower the x_j, the greater the superiority.

Note that the analysis above assumes the same value of W_{ai} for both schemes and thus ignoring the additional cost of redundant requests. So, the advantage of RRS over RDS depends on specific conditions.

In this paper, we focus on MDS codes but we also study the latency of non-MDS codes in another version of our work [24]. Recently, many non-MDS codes have been proposed, such as Local Reconstruction Code [15], [25].

IV. PERFORMANCE EVALUATION

A. Simulation Setup



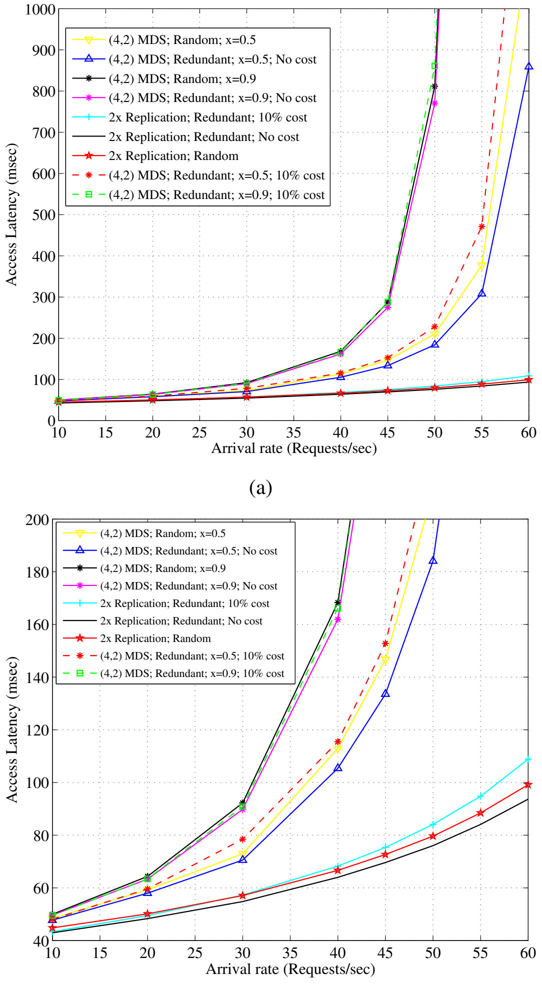
(b)

Figure 4. Latency performance of k-access read requests in (4,2) MDS and 2x replication systems.

We simulate systems with (n, k) MDS coding and $(c, 1)$ Replication, respectively. We use different parameters of n, k and c, and find that they exhibit similar results. We assume all read requests arrive as a Poisson process with parameter λ to an n-node structure in our simulations. Besides, we suppose both k-access and direct read requests possess the same priority. We take the access latency over 1000 sample paths for each experiment.

B. Simulation Results and Analysis

Since the measurements over Amazon S3 in [3] indicates that downloading time can be accurately approximated as an exponentially distributed random variable. As with previous work [9], [26], we conduct experiments under exponential service times with parameter μ for each storage node. From the results of real traces over Amazon S3 [14], we observe that the average smallest service time of reading a 1MB file is around 40 msec, and we can regard it as the actual



(a)

(b)

Figure 5. Latency performance of direct read requests in (4,2) MDS and 2x replication systems.

average service time with the queueing delay removed. Consequently, we suppose the size of each fragment is 1MB and set $\mu = 25$ requests/sec in our simulations.

First, we examine the latency performance of coding ((4,2) MDS) and replication (2x) in the case when there are only k-access reads, which is widely discussed by previous work [3], [4], [5], [9]. Since we also consider the additional cost of RRS, we get more practical results. As shown in Fig. 4 (a) and (b), where (b) is the expanded part of (a), corresponding to small access latency, we can observe that, for any k-access read, the (4,2) MDS invariably outperforms 2x Replication in terms of access latency under different conditions. It is also noted that the access latency under RRS is strictly smaller than that under RDS when no additional cost is considered. The two results above are in line with previous work. While if we consider a 10% additional cost, the latency performance under RDS is superior to that under RRS. This observation validates our analysis in the last section. Therefore, some theoretical results [9] without considering the additional cost of redundant requests are overly optimistic and may not hold in practice.

In addition, we study the latency performance of coding ((4,2) MDS) and replication (2x) in another case when there are only direct reads in the systems. To facilitate understanding, we suppose the number of direct read requests to the two data fragments is the same. As illustrated in Fig. 5 (a) and (b), where (b) is the expanded part of (a), corresponding to small access latency, we can see that 2x Replication delivers lower access latency compared with (4,2) MDS code no matter what the direct read fraction x is. This result is totally different from that of k-access read requests. The reason is that Replication can easily balance the load of direct reads between different copies, while MDS code can only transfer part of the direct reads via degraded reads which will inevitably increase the whole workload in the system, leading to higher access latency. It can also be observed that (4,2) MDS code with $x = 0.9$ suffers much higher access latency compared with (4,2) MDS code with $x = 0.5$. It is not surprising since when $x = 0.9$, 90% direct read requests go to the two data fragments and only 10% of them are transferred to parity fragments with degraded reads. However, the system can reasonably transfer around half of the direct reads to parity fragments when $x = 0.5$. That is, MDS code with $x = 0.5$ can achieve better load balancing compared with that with $x = 0.9$, thus reducing access latency, which also verifies our analysis in the last section. Similar to the results in Fig. 4, the effectiveness of RRS on reducing access latency compared with that of RDS also depends on the additional cost of redundant requests.

Then, in the following simulations, we use real service time traces from Amazon S3 shared by the authors of [14]. These traces are for read files of 1MB in size from an S3 bucket, located in North California region.

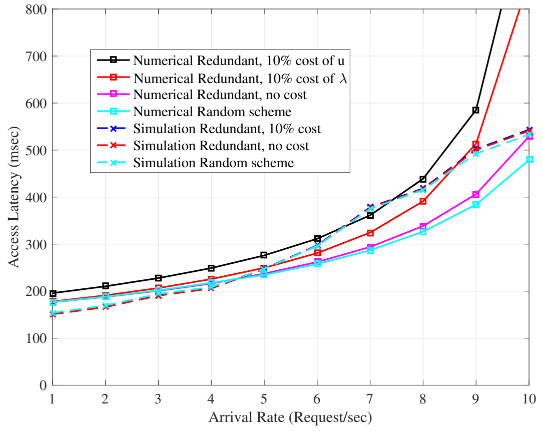


Figure 6. Latency performance of direct reads in (6,3) MDS storage systems.

We first compare the numerical and simulation results of direct reads in (6,3) MDS-coded storage systems. As illustrated in Fig. 6, the dotted lines are simulation results and the solid lines are numerical results. Numerical results can be obtained from Theorems 1 and 2 and Corollary 1, and we assume that the queueing delay of each storage node is that of an $M/M/1$ queue and the service rate $\mu = 6$. The arrival rates (x-axis) are the total requests to a codeword and different data nodes have different request arrival rates. We can observe that, for both numerical and simulation results, the random scheme achieves lower latency than the redundant scheme when the request arrival rate is high and the redundant scheme without incurring any additional cost can realize lower latency than the random scheme when the request arrival rate is low. Besides, with 10% additional cost, the redundant scheme performs poorly compared with the random scheme in terms of latency. We can also find that, compared with simulation results when there is 10% additional cost, our proposed numerical model with 10% additional cost of request arrival rate, i.e., increasing the system load by 10%, can better capture the reality than the previous model [23] with 10% additional cost of service rate. All the above demonstrate that our performance model works well and can easily obtain the latency performance of direct reads.

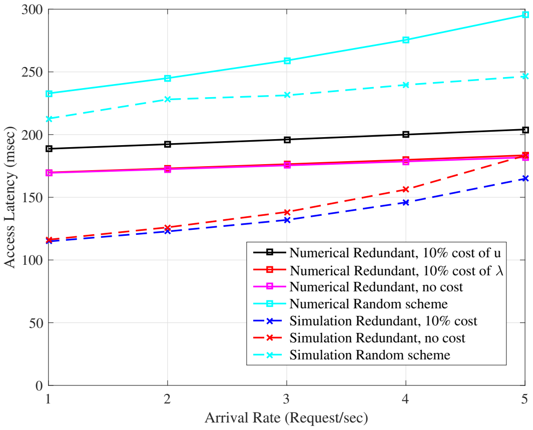


Figure 7. Latency performance of k-access reads in (6,3) MDS storage systems.

Next, let us compare the numerical and simulation results of k-access reads in (6,3) MDS-coded storage systems. As illustrated in Fig. 7, the dotted lines are simulation results and the solid lines are numerical results from Theorems 1 and 2 and Corollary 1. We can obtain the numerical results with the same set of curves as in Fig. 6. In this case, there are both direct reads and k-access reads in the systems at the same time, while we focus on the latency performance of k-access reads. The arrival rates on the x-axis are the total requests to a codeword, and different data nodes have different request arrival rates. Similar to Fig. 6, both numerical and simulation results deliver similar conclusions. It is noted that RRS achieves lower latency than RDS in this case. Besides, with 10% additional cost, the redundant scheme performs worse than that without. All the above demonstrate that our performance model works well and can easily obtain the latency performance of k-access reads.

V. Conclusion and Future Work

Much of the research regarding latency performance only discusses k-access read requests while we also study direct read requests. In this paper, we propose a practical model to analyze latency performance in distributed storage systems. In this model, we consider all kinds of read tasks, including direct, degraded and k-access reads, and we also account for more practical issues, such as the additional cost of redundant requests. We demonstrate how the new model can be easily used for different codes and dispatch schemes and point out that the effectiveness of MDS code and Replication on reducing access latency is different for direct read and k-access read requests. We also show that some previous work may be overly optimistic on the effectiveness of RRS at improving latency performance considering the additional cost in practice. We validate our results through simulations with real service times traces from Amazon S3.

Although in the paper, we focus on read requests, our model can also be extended to study write requests. Besides, as repair requests may also impact latency performance [2], [13], including repair requests in our model will probably make our model more general. In this paper, we divide direct read requests into direct and degraded read tasks. It would be interesting to study how the fraction x of direct read tasks in this model can influence latency performance.

References

[1] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google file system," in *ACM SIGOPS Operating Systems Review*, vol. 37, no. 5. ACM, 2003, pp. 29–43.

[2] Q. Shuai, V. O. K. Li, and Y. Zhu, "Performance models of access latency in cloud storage systems," in *Fourth Workshop on Architectures and Systems for Big Data*, 2014.

[3] S. Chen, Y. Sun, U. C. Kozat, L. Huang, P. Sinha, G. Liang, X. Liu, and N. B. Shroff, "When queueing meets coding: Optimal-latency data retrieving scheme in storage clouds," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 1042–1050.

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

[5] N. B. Shah, K. Lee, and K. Ramchandran, "The MDS queue: Analysing the latency performance of erasure codes," in *Information Theory (ISIT), 2014 IEEE International Symposium on*. IEEE, 2014, pp. 861–865.

[6] Q. Shuai and V. O. K. Li, "Delay performance of direct reads in distributed storage systems with coding," in *Proceedings of the 17th International Conference on High Performance Computing and Communications (HPCC)*. IEEE, Aug 2015, pp. 184–189.

[7] A. Vulimiri, O. Michel, P. Godfrey, and S. Shenker, "More is less: reducing latency via redundancy," in *Proceedings of the 11th ACM Workshop on Hot Topics in Networks*. ACM, 2012, pp. 13–18.

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

[9] G. Joshi, Y. Liu, and E. Soljanin, "On the delay-storage trade-off in content download from coded distributed storage systems," *IEEE Journal on Selected Areas in Communications*, vol. 32, no. 5, pp. 989–997, 2014.

[10] Q. Shuai and V. O. K. Li, "Reducing delay of flexible download in a coded distributed storage system," in *Proceedings of Global Communications Conference (GLOBECOM)*. IEEE, Dec 2016.

[11] G. Liang and U. C. Kozat, "TOFEC: Achieving optimal throughput-delay trade-off of cloud storage using erasure codes," in *INFOCOM, 2014 Proceedings IEEE*. IEEE, 2014, pp. 826–834.

[12] J. McCullough, J. Dunagan, A. Wolman, and A. C. Snoeren, "Stout: An adaptive interface to scalable cloud storage," in *USENIX Annual Technical Conference*, 2010, pp. 47–60.

[13] Q. Shuai and V. O. K. Li, "HTSC and FH_HTSC: XOR-based codes to reduce access latency in distributed storage systems," *Journal of Communications and Networks*, vol. 17, no. 6, pp. 582–591, Dec 2015.

[14] 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.

[15] C. Huang, H. Simitci, Y. Xu, A. Ogus, B. Calder, P. Gopalan, J. Li, S. Yekhanin *et al.*, "Erasure coding in Windows Azure storage," in *USENIX ATC*, 2012.

[16] Q. Shuai and V. O. K. Li, "Flexible download time analysis of coded storage systems," in *Proceedings of the 9th ACM International on Systems and Storage Conference*. ACM, 2016, p. 26.

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

[18] O. Khan, R. Burns, J. Plank, W. Pierce, and C. Huang, "Rethinking erasure codes for cloud file systems: Minimizing I/O for recovery and degraded reads," in *Proc. of USENIX FAST*, 2012.

[19] M. Sathiamoorthy, M. Asteris, D. Papailiopoulos, A. G. Dimakis, R. Vadali, S. Chen, and D. Borthakur, "Xoring elephants: Novel erasure codes for big data," in *Proceedings of the 39th international conference on Very Large Data Bases*. VLDB Endowment, 2013, pp. 325–336.

[20] K. V. Rashmi, N. B. Shah, and P. V. Kumar, "Optimal exact-regenerating codes for distributed storage at the msr and mbr points via a product-matrix construction," *IEEE Transactions on Information Theory*, vol. 57, no. 8, pp. 5227–5239, 2011.

[21] M. Foley, "High availability HDFS," in *28th IEEE Conference on Massive Data Storage, MSST*, vol. 12, 2012.

[22] J. Brutlag, "Speed matters for Google web search," *Google. June*, 2009.

[23] N. B. Shah, K. Lee, and K. Ramchandran, "The MDS queue: Analysing latency performance of codes and redundant requests," Tech. Rep., 2013.

[24] Q. Shuai and V. O. K. Li, "A general model of latency performance in distributed storage systems," Department of Electrical and Electronic Engineering, The University of Hong Kong, Tech. Rep. No.TR-2015-03, April 2015.

[25] P. Gopalan, C. Huang, H. Simitci, and S. Yekhanin, "On the locality of codeword symbols," 2012.

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