Title	Performance models of access latency in cloud storage systems
Author(s)	Shuai, Q; Li, VOK; Zhu, Y
Citation	The 4th Workshop on Architectures and Systems for Big Data (ASBD 2014) held in conjunction with The 41st International Symposium on Computer Architecture (ISCA 2014), Minneapolis, MN., 15 June 2014.
Issued Date	2014
URL	http://hdl.handle.net/10722/204118
Rights	Creative Commons: Attribution 3.0 Hong Kong License

Performance Models of Access Latency in Cloud Storage Systems

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

Yixuan Zhu
Department of Electrical and Electronic Engineering
The University of Hong Kong
Hong Kong, China
Email: yxzhu@eee.hku.hk

***Abstract*—Access latency is a key performance metric for cloud storage systems and has great impact on user experience, but most papers focus on other performance metrics such as storage overhead, repair cost and so on. Only recently do some models argue that coding can reduce access latency. However, they are developed for special scenarios, which may not reflect reality. To fill the gaps between existing work and practice, in this paper, we propose a more practical model to measure access latency. This model can also be used to compare access latency of different codes used by different companies. To the best of our knowledge, this model is the first to provide a general method to compare access latencies of different erasure codes.**

***Index Terms*—Access latency, erasure codes, MDS property, degraded reads, computation cost.**

I. Introduction

Access latency indicates the availability of storage systems and it can be measured as the average time taken to read data from storage nodes. Access latency is very important in cloud storage systems as it greatly impacts user experience. For example, Google found that users performed fewer searches after 4 to 6 weeks because of a 400 millisecond additional delay (up to 0.74% fewer searches after the delay has been implemented for 4 to 6 weeks) [1]. In general, node availability with the 3-replica strategy is higher than that with erasure coding [2] due to the extra complexity of coding. Recently, a few papers have studied the effectiveness of erasure codes at reducing access latency. By queueing-theoretic analysis of coded systems, [3], [4] proposed algorithms and argued that erasure codes can reduce access latency. Based on fork-join queues for parallel processing, [5] generalized the (n,n) fork-join system and found bounds on its mean response time. [6] argued that redundant requests in the context of the wide-area Internet can help reduce latency. A theoretical analysis in [7] shows that sending redundant requests can help reduce access latency in a coded storage system with maximum distance separable (MDS) property. Despite such efforts, an accurate performance model of access latency in cloud storage systems is still lacking.

First, almost all the above papers assume that each request to the storage system needs to access at least k storage nodes. But in practice, the storage code deployed is usually a systematic code, which means that one copy of the data exists in uncoded form [8], to facilitate applications such as keyword searching. Besides, in some systems such as Windows Azure Storage (WAS), not until the files reach a certain size (e.g., 3GB), will they be a candidate for erasure coding [9]. Although MDS codes can recover the whole data with any k out of n storage nodes, in a storage system with erasure codes, we will not just divide any file into k fragments no matter how small it is. We will combine many files into a fixed size and then divide them into k data fragments and add parity fragments to increase fault tolerance. Considering that requests usually do not need all the 3GB content, even in cloud storage systems with erasure coding, most reading requests just need to read data from one storage node in practice.

Second, in cloud storage systems, other than the usual data retrieval, repairing failures is a frequent operation [10]. Cloud storage systems such as Google file system (GFS), Amazon S3 and WAS, assemble massive amounts of unreliable hardware. Facebook's Hadoop distributed file system (HDFS) needs to transfer around 180TB data across racks per day for recovery operations and there are many high repair rate periods every day [11]. As shown in Fig. 1 (we will show how to get this figure in a later section), in cloud storage systems, for a certain reading requests arrival rate λ_1, the access latency will greatly increase with the increase of repairing requests arrival rate λ_2. Especially when λ_2 is large, a little increase of λ_1 or λ_2 may dramatically increase the access latency of reading requests. So, when we study access latency in cloud storage systems, we must consider the impact of repairing failed fragments on the access latency of reading requests for the data. However, to the best of our knowledge, this has never been mentioned in the previous work on access latency.

Third, while degraded reads [12], [13] are common in cloud storage systems, they are ignored by 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 reading request, and we need to reconstruct that storage node with the data from other nodes to meet the requirement of this new reading request. Thus a new reading request at an unavailable node generate degraded reads at other

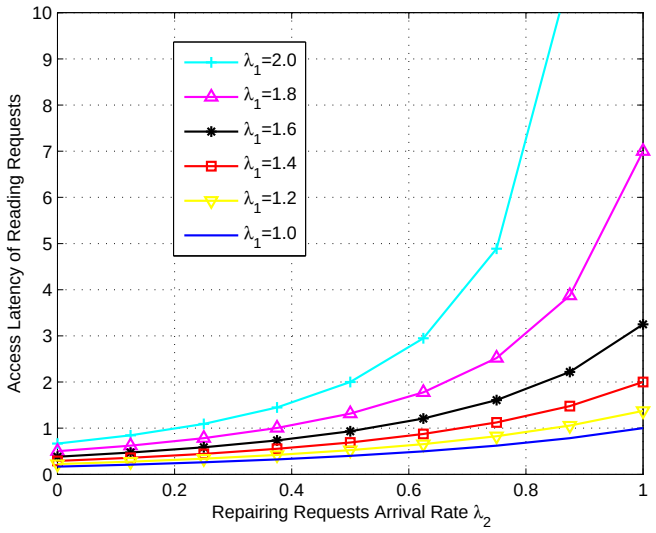

Fig. 1. The access latency of reading requests varies with different repairing request arrival rates.

nodes. Degraded reads are similar to failure repair but the former are due to transient unavailability while the latter is permanent data loss which has to be recovered. Degraded reads will influence the access latency as much as repairs and must be included in the access latency analysis.

Fourth, algorithms and models in previous work like [3] only compare the access latency of MDS erasure codes with the replication strategy and cannot compare the latencies of different erasure codes.

Finally, [3] further assumes that the cost of removing unfinished jobs is negligible. This is usually not the case, and servers may stay idle for a time before they can remove the unfinished jobs [7]. With redundant requests, MDS codes may help decrease access latency but only in some specific circumstances under ideal conditions. Perhaps, this is why GFS, WAS and Facebook HDFS still use 3-replica strategy by default [9], [11].

Our assumptions. In our new model, we assume that 1) in line with most previous work, the storage systems are homogeneous and the failures of different storage nodes are independent; 2) most reading requests just need to access one storage node and if some reading requests need to read from more than one node, degraded reads may be used; 3) repair requests and degraded reads are quite common.

Our Contributions. In this paper, we propose a new queueing model which accounts for the redundancy of erasure codes and the impact of degraded reads and repair requests to better measure the access latency in cloud storage systems. To the best of our knowledge, this model is the first to give a general way to measure and compare the access latency of different erasure codes.

The remainder of this paper is organised as follows. In Section II we propose a new model to measure access latency and explain how it works. In Section III we show numerical results to compare access latencies of different codes with our new model. Finally, in Section IV, we conclude and discuss some open questions.

II. System Model

To study the access latency in storage systems, models in [3], [5], [7], [14] try to model the data retrieval process. Although these models are already very complicated, they still suffer from many unrealistic assumptions. One key problem is that cloud storage systems are very complicated and contain many storage nodes, that it is almost impossible to keep track of all these storage nodes' concurrent actions on different requests simultaneously with one queueing model. We overcome this complexity by decomposing the whole storage system into individual storage nodes and analyzing one of them. Since the system is assumed to be homogeneous, the access latency performance of the whole system can be estimated from the access latency performance of an arbitrary data storage node.

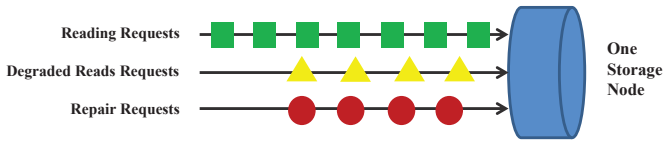

Fig. 2. Different requests to one storage node in cloud storage system.

In cloud storage systems, reading requests to a storage node are divided into two types. One is just direct reading from that storage node, and the other is from degraded reads (meaning the reconstruction of that node from other storage nodes because of transient unavailability of that node). As shown in Fig. 2, for any one storage node, requests in the queue are divided into three groups, reading requests, degraded reads requests and repair requests. Although degraded reads are resulted from reading requests on other storage nodes, they have the same importance with direct reading requests since they all influence user experience. In our model, we merge degraded reads and reading requests into one general reading requests queue, as shown in Fig. 3. General reading requests arrival rate is λ_1 and repair requests arrival rate is λ_2, μ_1 and μ_2 respectively are the service rate of reading requests and repair requests. Although writing is also a routine process in cloud storage systems, since we usually use append-only method to add new content or update some content in massive distributed storage systems [15], writing requests do not much influence the latency of the system. So we just consider the reads, degraded reads and repair requests here.

We model the general reading and repair requests as a head-of-the-line (HOL) priority queueing system. Since we try our best to guarantee no data loss in cloud storage systems, repair requests should have higher priority than general reading requests. Therefore, repair requests always queue in front of reading requests. But in their respective groups, requests follow the rule of first-come-first-served.

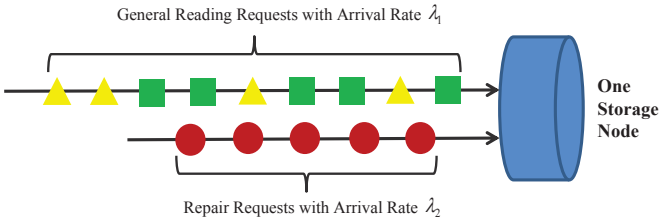

Fig. 3. General reading requests and repair requests to one storage node.

For the HOL priority queueing system, we can compute the access latency of general reading requests as $w_1 = \frac{w_0+\rho_2 w_2}{1-\rho_1-\rho_2}$, where $w_0 = \frac{\lambda_1 E(x_1^2)}{2} + \frac{\lambda_2 E(x_2^2)}{2}$ is the average access latency for both general reading requests and repair requests without considering priorities, $w_2 = \frac{w_0}{1-\rho_2}$ is the actual access latency of repair requests in HOL priority queueing systems, $\rho_1 = \frac{\lambda_1}{\mu_1}$, $\rho_2 = \frac{\lambda_2}{\mu_2}$, x_1 and x_2 are the service times of general reading requests and repair requests, respectively [16]. As in most of the previous work, we assume Poisson arrivals and exponential service times for both general reading and repair requests. So, we have $\sigma_{x_1}^2 = \frac{1}{\mu_1^2}$, $E(x_1) = \frac{1}{\mu_1}$ and $\sigma_{x_2}^2 = \frac{1}{\mu_2^2}$, $E(x_2) = \frac{1}{\mu_2}$. Since $E(x^2) = \sigma_x^2 + E(x)^2$, we can compute the access latency of general reading requests w_0 with the parameters λ_1, λ_2, μ_1 and μ_2. If we set parameters $\mu_1 = 3$ and $\mu_2 = 3$, we can get the numerical result in Fig. 1, which shows the impact of λ_1 and λ_2 on access latency.

We know that in such an HOL priority queueing system, λ_1, λ_2, μ_1 and μ_2 can all influence the access latency of general reading requests. For one storage node, reading requests arrival rate usually depends on data content and is uncorrelated with the coding technique used. While degraded reads and repair requests are highly correlated with the codes used in the storage system. That is, general reading requests, including both reads and degraded reads, with arrival rate λ_1 and repair requests, with arrival rate λ_2 are both correlated with codes used in the system. Service rate of general reading requests μ_1 and repair requests μ_2 usually have negative correlation with the complexity of the encoding and decoding processes. However, it is very hard to provide a quantitative analysis on how coding complexity can impact μ_1 and μ_2 of different coding storage systems. Here we focus on the influence of code types on λ_1 and λ_2 and show how they will influence access latency in cloud storage systems.

To compare the access latency of different erasure codes, we first compute λ_1 and λ_2 for systems with different coding methods and then use the priority queueing model above to get the access latency of different codes.

Method to calculate λ_1. In a storage system, suppose reading requests arrival rate for any one storage node is λ_1'. Degraded reads requests arrival rate for one node is positively correlated with λ_1' and the number of other storage nodes it is connected to, to assist in its reconstruction. Suppose a fraction x of all reading requests are direct reads, while 1-x of them are degraded reads. So the direct reading requests to that storage node is $x\lambda_1'$. Suppose that this storage node is connected with n other nodes, and the probability that this storage node join the reconstruction of any of the n nodes is p. Then we can get the degraded reads requests for that storage node is $(1-x)np\lambda_1'$. We can get the general reading requests arrival rate for that node as $\lambda_1 = x\lambda_1' + (1-x)np\lambda_1'$.

Method to calculate λ_2. In cloud storage systems, the probability of one failure is usually much higher than that of more than one failure. For example, in Facebook warehouse cluster, the proportion of single block recovery is as high as 98.08% [11]. So in this model, to compute λ_2, we just assume single failure.

Here we give a series of steps to calculate the relative value of λ_2 for different codes to be compared.

1) Choose one of the codes, say Code 1. In Code 1, suppose for any storage node, original repair requests arrival rate is λ_2'. Then use λ_2' as the unit value to compute the values of other codes.

2) In Code 1, arbitrarily choose one data storage node, if the number of nodes in Code 1 connected to this node is n1, and for another code such as Code 2, also arbitrarily choose one data storage node. If the number of nodes in Code 2 connected to that node is n2, then we can adjust λ_2' of Code 2 to $\frac{n_2}{n_1}\lambda_2'$. This is because the more the number of nodes connected to a storage node in a system, the higher the probability of that storage node to be accessed to help repair some failed nodes.

3) Suppose in all these codes, the probability of any storage node to join the repair of other nodes connected to it is p_i, $i = 1, 2, ...$, which is different for different codes. Then we can get the repair requests arrival rate of Code 1 as $\lambda_2 = p_1\lambda_2'$. For Code 2, $\lambda_2 = p_2\frac{n_2}{n_1}\lambda_2'$. Similarly, we can get the λ_2 of all other codes.

We need to calculate the value p_i according to the specific structure of codes to be compared. When we calculate p_i, we need to consider the redundancy of erasure codes, the different conditions of local and global parity fragments for Local Reconstruction Codes [9] and so on.

III. Numerical Results and Analysis

We will start with the access latency comparison between the 2-replica strategy and (4,2) MDS codes (this example is used in [3], [4]). As shown in Fig. 4, for each method, there are four storage nodes.

Suppose reading requests arrival rate to some content in one storage node is λ_1'. Since in the 2-replica strategy, two storage nodes have the same content, reading requests arrival rate to each node is $0.5\lambda_1'$. It is easy to get $\lambda_{1\_2-replica} = x0.5\lambda_1' + 1(1-x)0.5\lambda_1' = 0.5\lambda_1'$. Since any 2 nodes out of MDS(4,2) codes can reconstruct the node of transient unavailability, $p_{MDS(4,2)} = \frac{2}{3}$ and n=3. That is $\lambda_{1\_MDS(4,2)} = x\lambda_1' + 3(1-x)\frac{2}{3}\lambda_1' = (2-x)\lambda_1'$, where $0 \le x \le 1$. Whatever x is, we always get $\lambda_{1\_2-replica} < \lambda_{1\_MDS(4,2)}$, and the smaller x is, the bigger the difference between the two values.

Similarly, with the methods in the last section, we set the 2-replica strategy's original repair requests arrival rate λ_2' as

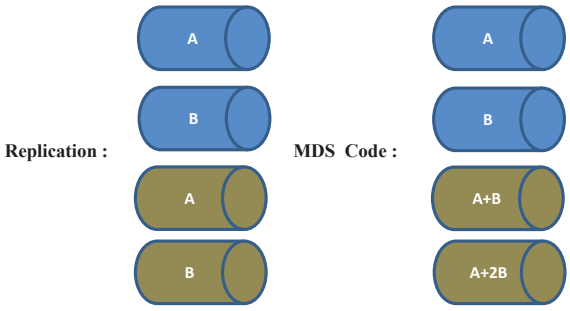

Fig. 4. 2-replica strategy storage and (4,2) MDS code storage.

unit value. We can get $\lambda_{2\_2-replica} = \lambda_2'$ and $\lambda_{2\_MDS(4,2)} = 3\frac{2}{3}\lambda_2' = 2\lambda_2'$.

If we set $\lambda_2' = 0.1$, $x = 0.9$, $\mu_1 = 2$ *and* $\mu_2 = 2$, with HOL priority queueing we can get the access latency of 2-replica and MDS(4,2) as shown in Fig. 5.

Obviously, the access latency of the 2-replica strategy is much lower than the MDS(4,2) Code. Actually, we have not considered the encoding and decoding complexity of MDS(4,2) Code compared with the 2-replica strategy yet. If we do, the difference of access latency between the 2-replica strategy and the MDS(4,2) Code will be even bigger.

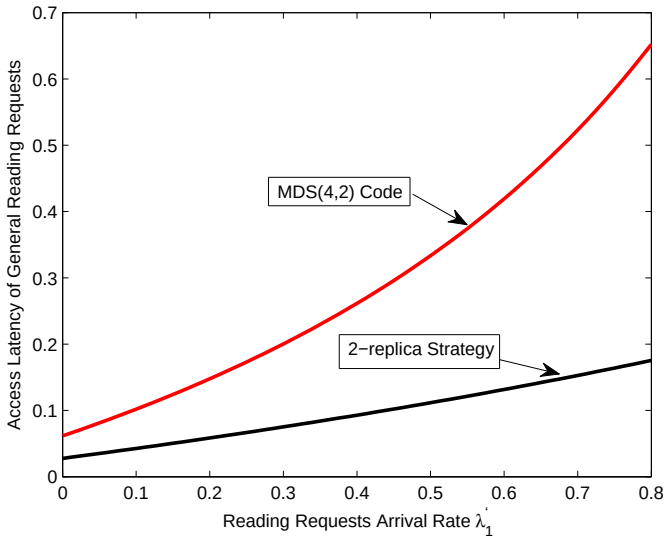

Fig. 5. Access latency comparison between 2-replica and MDS(4,2).

The result here is different from that in [3], [4]. They assume each request from users needs to read the whole coded data and to access at least k storage nodes. But as discussed in the introduction section, their models just consider some special cases under some ideal conditions, while our model considers more practical cases. Till now, GFS, WAS and Facebook HDFS still use 3-replica strategy by default [9], [11] to reduce access latency, especially for data requiring frequent retrievals. Our conclusion here is in line with industrial preference. We believe our model can better measure the access latency in practice for different storage strategies.

Next, we show how our model can help measure and compare access latency for different erasure codes. We will compare some popular codes, namely, Local Reconstruction Code (LRC)(12,2,2), LRC(6,2,2), Locally Repairable Codes (LRCs)(10,6,5), Reed-Solomon (RS) Code(6,3), RS(10,4) and RS(12,4). In particular, LRC is used in WAS in Microsoft [9], RS Code(6,3) is used in GFS II [17], [18], RS Code(10,4) is used in in HDFS in Facebook [19] and LRCs(10,6,5) are used in HDFS-Xorbas.

With our model in the last section, the calculations of λ_1 and λ_2 for RS codes are similar to the MDS(4,2) Code above. Suppose reading requests arrival rate to any storage node is λ_1', we can easily get $\lambda_{1\_RS(6,3)} = x\lambda_1' + 8(1-x)\frac{6}{8}\lambda_1' = (6-5x)\lambda_1'$, $\lambda_{1\_RS(10,4)} = x\lambda_1' + 13(1-x)\frac{10}{13}\lambda_1' = (10-9x)\lambda_1'$, $\lambda_{1\_RS(12,4)} = x\lambda_1' + 15(1-x)\frac{12}{15}\lambda_1' = (12-11x)\lambda_1'$.

The tough part is to calculate λ_1 of LRCs(10,6,5) and LRC.

In LRCs(10,6,5), when any global parity node fails, other parity nodes will help repair [13]. For any data storage node, when one of the other nodes is temporarily unavailable or fails, the probability of that node joining the reconstruction is $p_{LRCs(10,6,5)} = \frac{1}{5}\frac{1}{2} + \frac{4}{5}1 = 0.9$. We can get $\lambda_{1\_LRCs(10,6,5)} = x\lambda_1' + 5(1-x)p_{LRCs(10,6,5)}\lambda_1' = (4.5 - 3.5x)\lambda_1'$.

In LRC, for any data storage node, when any global parity node is temporarily unavailable or fails, there is some probability for that data storage node to help reconstruction. When one of the other nodes in the same local group is temporarily unavailable or fails, that storage node will help reconstruction [9]. Specifically, for LRC(6,2,2), we can get $p_{LRC(6,2,2)} = \frac{3}{9}1 + \frac{2}{9}(\frac{1}{2}\frac{3}{4} + \frac{1}{2}(\frac{1}{2}\frac{2}{4} + \frac{1}{2}\frac{3}{4})) = 0.486$ and $\lambda_{1\_LRC(6,2,2)} = x\lambda_1' + 9(1-x)p_{LRC(6,2,2)}\lambda_1' = (4.375 - 3.375x)\lambda_1'$. For LRC(12,2,2), similarly, $p_{LRC(12,2,2)} = \frac{6}{15}1 + \frac{2}{15}(\frac{1}{2}\frac{6}{7} + \frac{1}{2}(\frac{1}{2}\frac{5}{7} + \frac{1}{2}\frac{6}{7})) = 0.51$ and $\lambda_{1\_LRC(12,2,2)} = x\lambda_1' + 15(1-x)p_{LRC(12,2,2)}\lambda_1' = (7.65 - 6.65x)\lambda_1'$.

Following the steps of the last section, we set the original repair request arrival rate λ_2' of RS(6,3) Code as unit value, then we can get $\lambda_{2\_RS(6,3)} = 0.75\lambda_2'$, $\lambda_{2\_RS(10,4)} = 1.25\lambda_2'$, $\lambda_{2\_RS(12,4)} = 1.5\lambda_2'$, $\lambda_{2\_LRCs(10,6,5)} = 0.56\lambda_2'$, $\lambda_{2\_LRC(6,2,2)} = 0.55\lambda_2'$, and $\lambda_{2\_LRC(12,2,2)} = 0.96\lambda_2'$.

If we set $\lambda_2' = 0.1$, $x = 0.9$, $\mu_1 = 2$ *and* $\mu_2 = 2$, with HOL priority queueing we can get the access latency of different erasure codes as shown in Fig. 6 and Fig. 7. Fig. 7 is the expanded part of Fig. 6, corresponding to small λ_1'.

From Fig. 6 and Fig. 7 we can see that among the compared codes, LRC(6,2,2) used in WAS can achieve the lowest access latency while the access latency of RS(10,4) used in Facebook and RS(6,3) used in Google are both relatively high. Although LRC(12,2,2) is a variation of RS(12,4) and they have the same storage overhead, LRC(12,2,2) achieves much lower access latency by transfering two global parity nodes to local parity nodes [9]. LRCs(10,6,5) has much lower access latency compared with RS(10,4), and is a variation of RS(10,4) by adding two local parity nodes and using a deterministic algorithm with exponential complexity in the construction of code coefficients [13]. LRC(6,2,2) has almost the same access latency compared with LRCs(10,6,5). However, the former has lower repair cost and higher storage overhead, while the latter

has relatively higher fault tolerance capability.

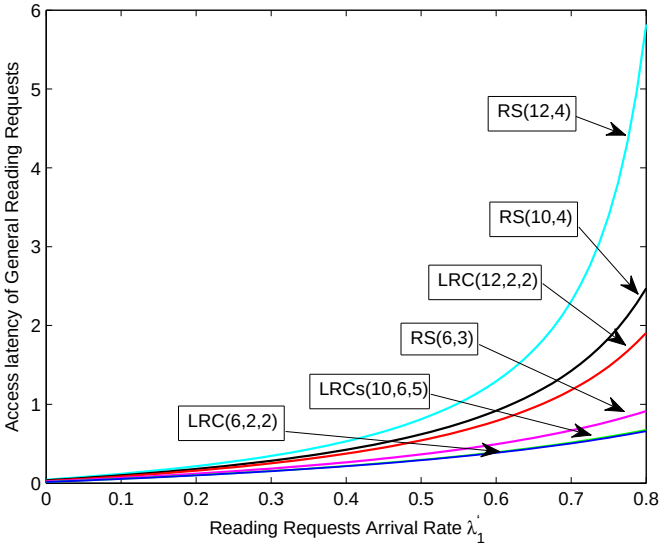

Fig. 6. Access latency comparison of different erasure codes.

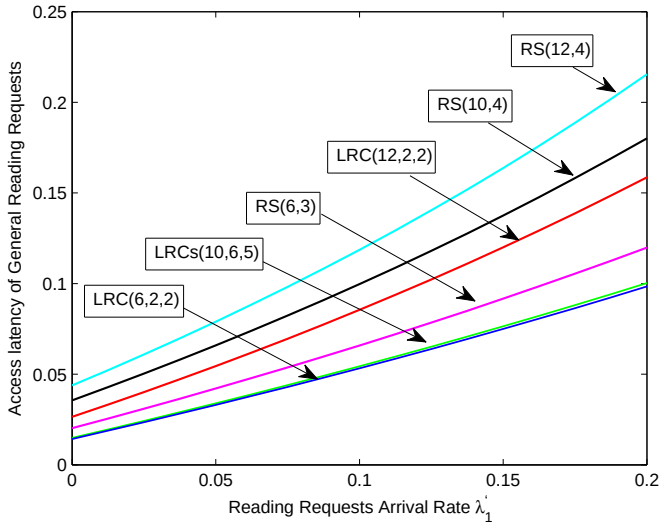

Fig. 7. Access latency comparison of different erasure codes.

IV. Conclusion and Future Work

In this paper, we propose a more realistic model to measure access latency and to the best of our knowledge, we are the first to provide a general model to measure and compare access latency between different erasure codes. Combining access latency with other performance metrics, we can better understand the advantages and disadvantages of different erasure codes and choose the code that matches our performance requirements best.

There are still some interesting open questions. First, we just consider the impact of one node's unavailability or failure on the access latency model since single recovery accounts for more than 98% storage repairs in cloud storage systems [11]. In the future, we may further generalize the model to consider conditions of more than one node recovery. Besides, since computational costs of different codes also impact access latency, it will be interesting to study such impact in the future.

References

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

[2] R. Rodrigues and B. Liskov, "High availability in DHTs: Erasure coding vs. replication," in *Peer-to-Peer Systems IV*. Springer, 2005, pp. 226–239.

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

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

[5] G. Joshi, Y. Liu, and E. Soljanin, "On the delay-storage trade-off in content download from coded distributed storage systems," *arXiv preprint arXiv:1305.3945*, 2013.

[6] 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.

[7] N. B. Shah, K. Lee, and K. Ramchandran, "When do redundant requests reduce latency?" *arXiv preprint arXiv:1311.2851*, 2013.

[8] 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.

[9] 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.

[10] 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.

[11] K. Rashmi, N. B. Shah, D. Gu, H. Kuang, D. Borthakur, and K. Ramchandran, "A solution to the network challenges of data recovery in erasure-coded distributed storage systems: A study on the Facebook warehouse cluster," in *Presented as part of the 5th USENIX Workshop on Hot Topics in Storage and File Systems*. USENIX, 2013.

[12] 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.

[13] 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.

[14] U. J. Ferner, M. Médard, and E. Soljanin, "Toward sustainable networking: Storage area networks with network coding," in *Communication, Control, and Computing (Allerton), 2012 50th Annual Allerton Conference on*. IEEE, 2012, pp. 517–524.

[15] K. Shvachko, H. Kuang, S. Radia, and R. Chansler, "The Hadoop distributed file system," in *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*. IEEE, 2010, pp. 1–10.

[16] L. Kleinrock, *Queueing systems: volume 2: computer applications*. John Wiley & Sons New York, 1976, vol. 82.

[17] A. Fikes, "Storage architecture and challenges," *Talk at the Google Faculty Summit*, 2010.

[18] D. Ford, F. Labelle, F. I. Popovici, M. Stokely, V.-A. Truong, L. Barroso, C. Grimes, and S. Quinlan, "Availability in globally distributed storage systems." in *OSDI*, 2010, pp. 61–74.

[19] D. Borthakur, R. Schmidt, R. Vadali, S. Chen, and P. Kling, "HDFS RAID," in *Hadoop User Group Meeting*, 2010.