Optimizing Content Retrieval Delay for LT-based Distributed Cloud Storage Systems

Haifeng Lu, Chuan Heng Foh, Yonggang Wen, and Jianfei Cai
School of Computer Engineering, Nanyang Technological University, Singapore 639798
Email: {lu0007ng, aschfoh, ygwen, asjfcai}@ntu.edu.sg

***Abstract*—Among different setups of cloud storage systems, fountain-codes based distributed cloud storage system provides reliable online storage solution through placing coded content fragments into multiple storage nodes. Luby Transform (LT) code is one of the popular fountain codes for storage systems due to its efficient recovery. However, to ensure high success decoding of fountain codes based storage, retrieval of additional fragments is required, and this requirement introduces additional delay, which is critical for content retrieval or downloading applications. In this paper, we show that multiple-stage retrieval of fragments is effective to reduce the content-retrieval delay. We first develop a delay model for various multiple-stage retrieval schemes applicable to our considered system. With the developed model, we study optimal retrieval schemes given the success decodability requirement. Our numerical results demonstrate that the content-retrieval delay can be significantly reduced by optimally scheduling packet requests in a multi-stage fashion.**

I. Introduction

Cloud storage systems provide a scalable online storage solution to end users who require flexible amount of storage space but do not wish to own and maintain storage infrastructure [1], [2]. Cloud storage systems consist of a collection of storage nodes that are connected through private or public Internet. A content such as a large file or a video is often fragmented and distributed in a set of storage nodes.

To offer high reliability and availability of storage services, redundancy of contents may be employed. Fragments of contents may be simply replicated and stored in different storage nodes to achieve redundancy. Erasure codes may be used for storage redundancy to improve storage efficiency of the system. In a distributed storage system with erasure coding, a large content is first divided into a number of fragments, say k fragments, usually of same size. These fragments undergo an encoding procedure specified by a particular erasure coding to produce an enlarged number of coded fragments, say n coded fragments, where $n-k$ is the storage redundancy. These coded fragments are distributed into different storage nodes for improved fault tolerance. In an optimal performing erasure code, the original content can be retrieved by obtaining any of the k coded fragments to reconstruct the content.

In general, there are two research efforts in designing an efficient distributed storage system with erasure codes for robustness. The first research effort falls on the design of enhanced erasure codes that improve system performance. In [3], Hafner presented Weaver codes as a highly fault tolerant erasure code for storage systems. By optimizing the Cauchy distribution matrix, Plank *et al.* introduced an enhanced Reed-Solomon codes for networked storage system [4]. In [5], Huang *et al.* proposed Pyramid codes where they explored employment of nested erasure codes as a single code. Oggier proposed Self-Repairing codes *et al.* [6] that achieves local decodability suitable for repairing of fragments in distributed storage systems.

Another research effort focuses on the design of storage system operational optimization based on existing erasure codes to achieve some specific optimal performance matrices. These operational designs may include storage redundancy overhead optimization, storage allocation and repair, optimal retrieval of storage, and others. In [7], Sardari *et al.* investigated the optimal storage allocation with limited storage budget considering Maximum Distance Separable (MDS) codes. Leong *et al.* studied storage allocation for high reliability [8]. Dimakis *et al.* introduced Regenerating codes for repairing of missing fragments considering linear network coding [9]. The first study investigating the relationship between storage system setup and retrieval delay is reported in [10] by Leong *et al.* where the authors studied relationship between storage allocation strategy and retrieval delay. Our research work falls under this effort of optimizing retrieval delay for a distributed storage system.

As high performance erasure codes often incur high complexity of computational, low computational complexity erasure codes appear advantageous in practical usage. Among various classes of erasure codes, fountain codes are a class that offers flexible redundancy due to its rateless property and relatively low computational complexity. One popular realization of fountain codes is Luby Transform (LT) codes [11]. LT codes enjoys relatively low computational complexity of $O(k\ln(k/\delta))$. However, to achieve this relatively low computational complexity, additional encoded fragments in the order of $O(\sqrt{k}\ln^2(k/\delta))$ are needed for LT decoder to achieve a successful decoding probability of $1-\delta$. The need for additional encoded fragments adds stress to the bandwidth and introduces delay in the retrieval.

In particular, to ensure a high probability of successful decoding probability, a storage collector operating LT decoder requires additional encoded fragments from the storage system. However, these additional encoded fragments add delay in storage retrieval, which is critical for content retrieval or downloading applications. Thus a tradeoff between the successful decoding probability and storage retrieval delay

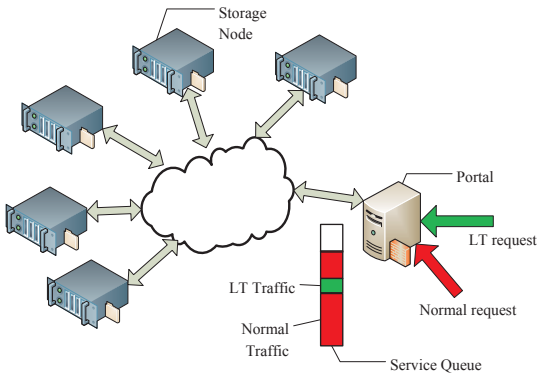


Fig. 1. System Architecture for a distributed data storage system.

exists. In this paper, we study this tradeoff and introduce a multiple stage storage retrieval scheme where we show that storage retrieval delay can be reduced without sacrificing the performance on decoding probability. We further demonstrate the optimal retrieval setup for the case of a 2-stage retrieval. With this optimal retrieval scheme, we believe it will benefit practical cloud storage systems through providing better QoS in terms of retrieval delay.

The remainder of this paper is organized as follows. In Section II, we introduce the system architecture and present the formal definition of the optimal retrieval scheme problem. Section III presents a rigorous analysis on the delay for different retrieval scheme together with simulation validation. We then demonstrate the tradeoff of the delay and probability of successful decoding in Section IV. Finally, some important conclusions are drawn in Section V.

II. System Architecture and Problem Statement

In this section, we first present a detailed description of a distributed data storage system, in which LT-encoded data fragments are spread out across a pool of storage nodes. Following that, we formulate an optimal file retrieval problem, which aims to minimize the retrieval delay by strategically scheduling packet retrieval requests.

A. System Architecture

We focus on a distributed data storage system (e.g., an community-based cloud storage system as in Tahoe-LAFS [12]) in which each storage node stores both LT fragments or packets[1] and normal packets for different contents. As illustrated in Fig. 1, the users can store and retrieve a file from the cloud storage system, via a dedicated *portal*. When storing a file into the cloud, the portal encodes the file with the LT code and spreads all the encoded fragments across different storage nodes for reliability. Upon receiving the file retrieval request, the portal sends a few packet retrieval requests to different storage nodes, and reassemble the list of received packets into the file and forward it to the user.

[1]Here we use the term *packet* interchangeably with *fragment* as we assume that an IP packet carries an LT encoded fragment in the system.

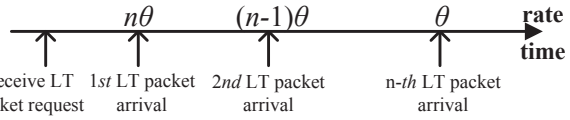


Fig. 2. Arrival process of LT encoded packets seen at the portal.

We assume that there is only one user who will request LT packets. The traffics generated by the requests from the rest users are treated as ambient traffics. Fig. 1 shows a snapshot of the considered system.

B. Traffic Model

In the portal, we consider a packet transmission queue with a fixed processing rate of r. The ambient traffic arrives with an exponential inter-packet time (i.e., an arrival rate of λ). The length of packets associated with the ambient traffic is L, which is a random variable with mean l and variance σ^2. Moreover, storage nodes transmit LT-encoded packets back to the portal, with an exponential inter-packet arrival time (or rate of θ). The length of LT encoded packets is a constant denoted by l_{LT}.

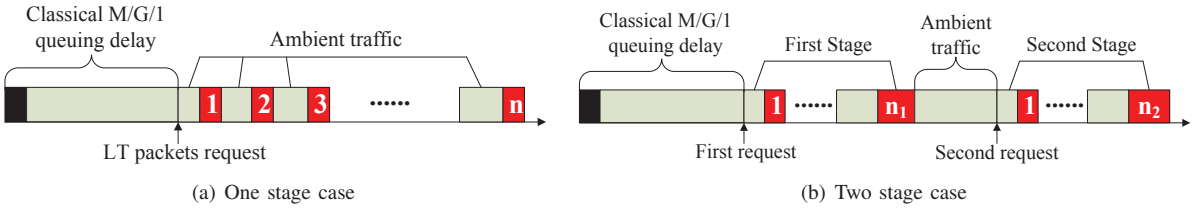
To recover a file of k original packets, the user may request n encoded packets from the portal in order to achieve successful decoding probability p. After the portal receives the request for n LT encoded packets, it retrieves these packets from the storage nodes. The requested n packets will then arrive at the portal at a random time which follows exponential distribution with a mean of $1/\theta$. According to [13], the arrival rate of i-th LT packets is $(n-i+1)\theta$. This arrival process of LT packets is illustrated in Fig. 2.

C. Problem Statement

In the considered distributed storage system, the bottleneck lies on the portal, because a large number of users can issue their file requests to the portal. In this paper, we focus on the file retrieval delay, defined as the duration between the time for portal receiving a LT-encoded file request and the time for the last LT packet leaves the portal. We aim to minimize the file retrieval delay by strategically scheduling the LT packet requests.

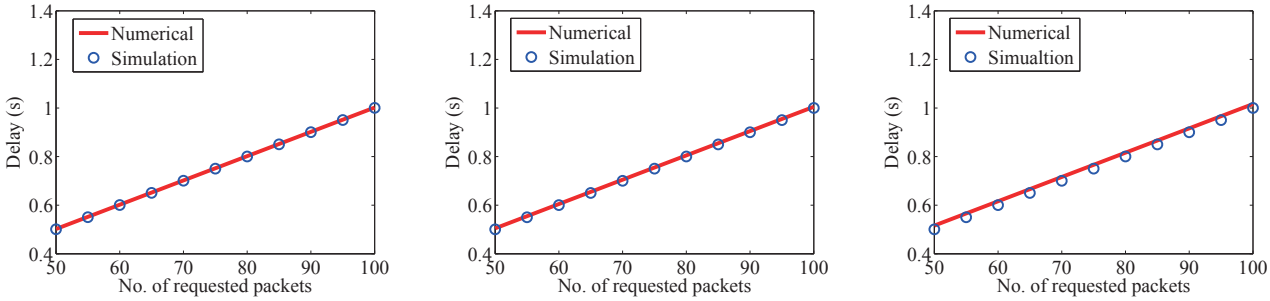
The optimal file retrieval problem is stated as follows. We assume a probabilistic file retrieval model. Specifically, in order to achieve the probability of successful decoding of p, the portal needs to request n LT encoded packets. Traditionally, the n packets are requested in one shot and the portal waits for all these packets to decode and retrieve the file. This scheme is referred to as an one-stage request scheme. In this paper, we propose to use a multiple stage request scheme to improve the file retrieval performance. Specifically, the portal can divide the request into t stages, each consisting of requests for $n_1, n_2, \cdots, n_t$ packets ($\sum_i n_i = n$). If the portal successfully decodes the file at stage m, it will stop requesting the rest $\sum_{i=m+1}^{t} n_i$ packets.

Intuitively, multiple stages of packet retrieval incurs additional delay in storage retrieval. Interestingly, we found that



(a) One stage case (b) Two stage case

Fig. 3. Components of delay.



(a) $\lambda = 10, \theta = 25$ (b) $\lambda = 30, \theta = 50$ (c) $\lambda = 60, \theta = 50$

Fig. 4. Delay in one stage request.

multiple stages of packet retrieval can outperform a single retrieval in terms of delay. This is because in multiple stages of packet retrieval, retrieving a just adequate number of fragments may already give sufficiently high success decodability which eliminates the need for further rounds of retrieval. In case that the decoding fails, a further round of retrieval may take place. By properly configure the multiple stage packet retrieval scheme, the gain in delay with a multiple stage retrieval outruns the loss due to additional rounds of retrieval. We demonstrate this by presenting an optimization of two-stage request scheme.

III. File-Retrieval Delay Analysis

We characterize the average file-retrieval delay for different request schemes in the presence of ambient traffic. We use computer simulation to validate our analytical results. In simulations, we set that the length of packets in the ambient traffic follows an exponential distribution. Although it is a simplified model, the insights obtained with this simple model can be applied to guide practical system design. Table I lists the default values of some parameters in the simulator. All the presented simulation results in the following sections are averaged from 1000 trails.

TABLE I
Default values in Simulation

Notation	Value
l	1024 bits
l_{LT}	1024 bits
r	100 kbps

A. Delay Analysis for One Stage Request Scheme

We first investigate the file-retrieval delay $D_1(n)$ for n LT packets in the one stage request case. This delay consists of two parts as shown in Fig. 3(a). The first part arises from the traffic arriving before the LT request and the second part arises from the traffic arriving after the LT request. The first part only contains ambient traffic. The first part is treated as a classical M/G/1 queue with delay denoted by W. The second part can be further divided into two sub-parts. Firstly, it contains a constant which is the transmission time of n LT packets, which can be derived as $\frac{nl_{LT}}{r}$. The rest is the time to process the ambient traffics which arrive at the portal during the inter-arrival time of each LT packet. This part is denoted by T. Thus, the delay $D_1(n)$ can be expressed as

$$D_1(n) = W + \frac{nl_{LT}}{r} + T. \tag{1}$$

Taking an expectation on both sides, we obtain

$$\begin{aligned} E(D_1(n)) &= E(W) + E(T) + \frac{nl_{LT}}{r} \\ &= \frac{\lambda(\sigma^2 + l^2)}{2r^2(1 - \frac{\lambda r}{l})} + \frac{\lambda l}{r}\sum_{i=0}^{n-1} E(\zeta_{i+1}^* - \zeta_i^*) + \frac{nl_{LT}}{r} \\ &= \Gamma + \frac{\lambda l}{r}\sum_{i=1}^{n}\frac{1}{i\theta} + \frac{nl_{LT}}{r} \\ &\simeq \Gamma + \frac{\lambda l}{r\theta}(ln(n) + 1) + \frac{nl_{LT}}{r}, \end{aligned} \tag{2}$$

where Γ is a constant, ζ_i^* is arriving time for the ith LT encoded packet and $\zeta_0^* = 0$.

In Fig. 4, we plot the numerical file retrieval delay, compared with the simulation results, as a function of the number of packets request, for various traffic loads. Notice that the numerical results match the simulation results well, verifying the applicability of our derived file-delay in (2). The results also demonstrate that the file-retrieval delay grows linearly with the number of packets requested.

B. Delay Analysis for Multiple Stage Request

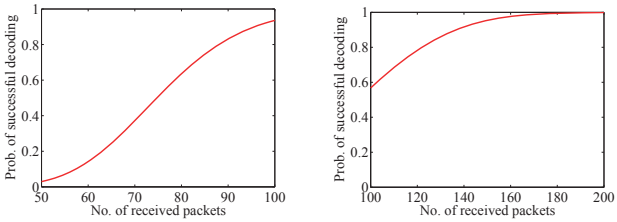
In this subsection, we first investigate the file-retrieval delay for the two stage request scheme. The packet arrival process is illustrated in Fig. 3(b).

Suppose in the first stage, the user requests n_1 LT encoded packets. The delay for the first stage is $D_1(n_1)$. After decoding these n_1 packets, if the user fails to decode the original file, it continues to request the n_2 packets. Here we assume that during the decoding process of n_1 packets, the transmission queue in the portal has already returned to the steady state. This assumption is reasonable since it does take some time for the user to determine if the n_1 LT-encoded packets are decodable. Thus, the delay for the second stage is identical to the first stage except for the number of encoded packets the user requests. As a result, the overall file-retrieval delay for the two stage request case is given by

$$\begin{aligned} D_2(n_1, n_2) &= f(n_1)D_1(n_1) \\ &+ (1 - f(n_1))(D_1(n_1) + D_1(n_2)), \end{aligned} \quad (3)$$

where $f(n)$ is the probability of successful decoding with n LT encoded packets. The determination of $f(n)$ is given separately in [14] and [15]. Here we adopt the model introduced in [15] due to its computational efficiency.

In Fig. 5, we plot the successful decoding probability as a function of the number of received packets when $k = 50$ and $k = 100$ respectively.



(a) $k = 50$ (b) $k = 100$

Fig. 5. The probability of successful decoding as a function of the number of received LT-encoded packets.

Using the results from Fig. 5, we can obtain the number of required packets for a targeted decoding probability. We plot in Figs. 6-7 the file-retrieval delay for the two-stage request scheme, as a function of the number of packets requested in the first stage, under different setting of k, n and traffic loads. Notice that the numerical results match the simulation results well, regardless of the traffic load. Moreover, in all results, the file-retrieval delay first decreases and then increases as the number of packets requested in the first stage increases. This consistent trend indicates a potential for file-retrieval delay minimization by choosing an appropriate number of packets requested in the first stage. We shall address this optimization in Section IV-A.

The result for the two stage request scheme can be generalized into arbitrary t stage request. Specifically, the delay expression for t stage request can be defined recursively,

$$\begin{aligned} D_t(n_1, n_2, \cdots, n_t) &= (1 - f(n_1))(D_1(n_1) \\ &+ D_{t-1}(n_2, \cdots, n_t)) \\ &+ f(n_1)D_1(n_1), \end{aligned} \quad (4)$$

where the initial values for D_1 and D_2 are defined in (1) and (3).

IV. Optimal Scheduling for Packet Retrieval

In this section, we will first investigate the optimal scheduling of packet requests to minimize the file-retrieval delay for two-stage request scheme, and then present a fundamental trade-off between the file retrieval delay and the probability of file decodability.

A. Optimal Request Scheme

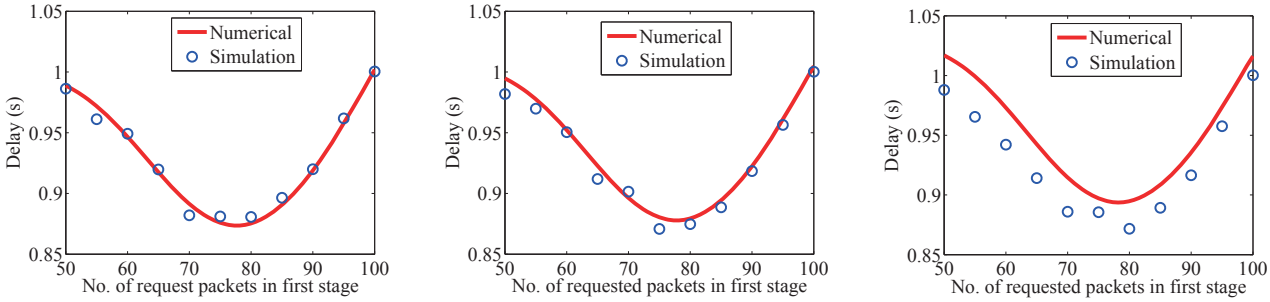
The results in Fig. 6-7 suggest an optimal two stage request scheme from (3). Solving the optimization problem yields the optimal number of packets requests in the first stage for a two-stage request scheme. In Fig. 8- 9, we plot, for the optimal two-stage request scheme, the ratio (n_1/n) between the number of requested packets in the first stage and the total number of packets needed, as a function of the decoding probability, for different traffic loads.

From these figures, we make an observation. We see that the optimal portion of packets requested in the first stage decreases as the target probability of successful decoding increases. This observation can be understood as follows. From the aspect of decodability probability, every packet contributes in the decoding process. From Fig. 5, we see an unequal contribution of each packet to the decodability probability. The contribution of an additional packet to the decodability probability is high when the probability stays at a lower level. As the probability progresses higher, the contribution of each additional packet drops.

Notice from Fig. 4 that delay in one stage request scheme increases linearly with the number of requested packets. Instead of requesting all the packets required by the target decodability probability at once, requesting a small portion of the number of packets needed in the first stage will generate a decoding probability that contributes most to the final decoding probability; while the marginal contribution from the second stage is relatively small. As a result, a two-stage request scheme helps keep the average delay lower than the one-stage request scheme.

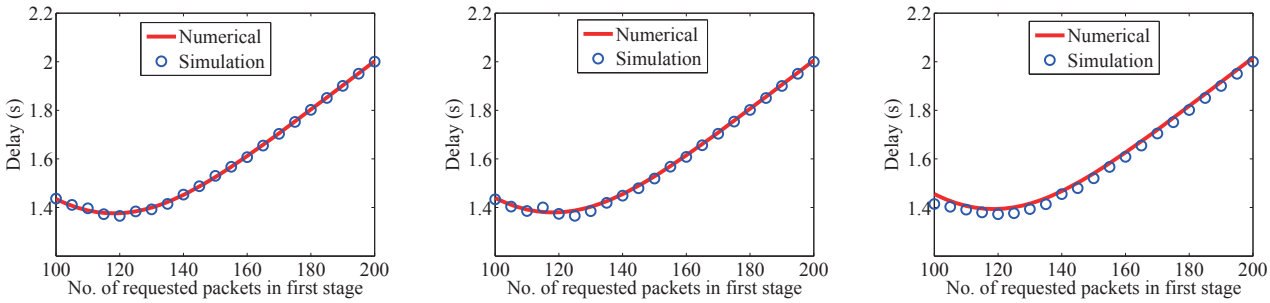
B. Delay-Decodability Tradeoff

We now investigate the fundamental tradeoff between the targeted decoding probability and the file-retrieval delay for different request schemes. This delay-decodability tradeoff is important since it helps users to make rational request scheme depending on the requirement of their applications. Using the results from previous two subsections, we plot the aforementioned trade-off for the one-stage request scheme and the two-stage request schemes, in Fig. 10-11, for different system loads.



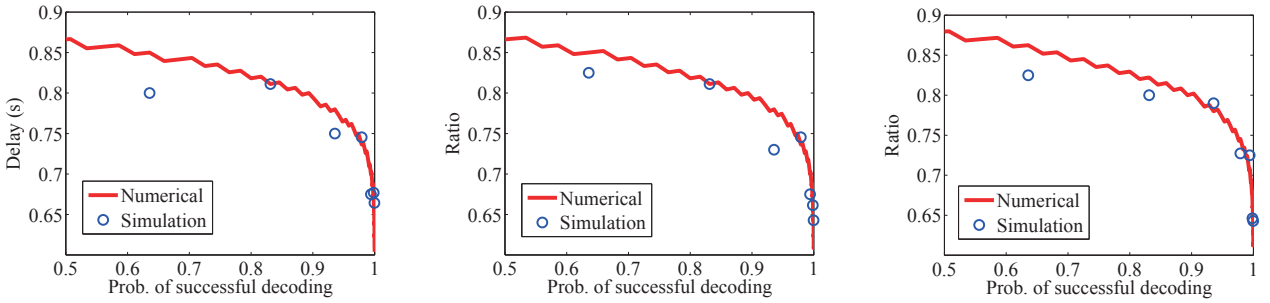
(a) $\lambda = 10, \theta = 25$ (b) $\lambda = 30, \theta = 50$ (c) $\lambda = 60, \theta = 50$

Fig. 6. Delay in two stage request when $k = 50$ and $n = 100$.



(a) $\lambda = 10, \theta = 25$ (b) $\lambda = 30, \theta = 50$ (c) $\lambda = 60, \theta = 50$

Fig. 7. Delay in two stage request when $k = 100$ and $n = 200$.



(a) $\lambda = 10, \theta = 25$ (b) $\lambda = 30, \theta = 50$ (c) $\lambda = 60, \theta = 50$

Fig. 8. Ratio of number of requested LT packets in first stage when $k = 50$.

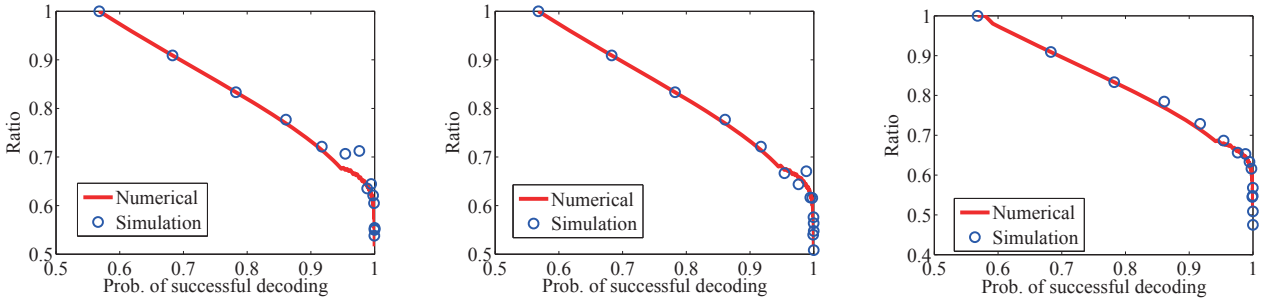
As observed in these figures, the delay for one-stage request increases sharply when the decoding probability crosses 90%. In comparison, for the two-stage request scheme, the delay is consistently lower. This advantage reflects the gain from being able to decode in the first stage with an adequate instead of excessive number of LT packets. Based on the reported delay-decodability tradeoff, user can choose an appropriate target of decodability probability, i.e. for some delay sensitive applications like stream video, a lower decodability probability can be adopted such that the average retrieval delay can be maintained under a certain threshold.

V. Conclusion

In this paper, we investigate the problem of optimal file retrieval under a distributed cloud storage system. The file is first LT-encoded and spread out into a list of distributed storage nodes. When retrieval, the portal schedules the packet request in a multi-stage manner, with an objective to minimize the average file retrieval delay. We developed an accurate model to characterize the average file-retrieval delay for different request strategies. Using this model, we derived an optimal two stage request scheme for a given decoding probability. Both simulation and numerical result confirm that this optimal scheme can reduce the average delay dramatically. We believe such delay-optimized retrieval scheme can benefit practical distributed cloud storage systems through providing better QoS in terms of retrieval delay.

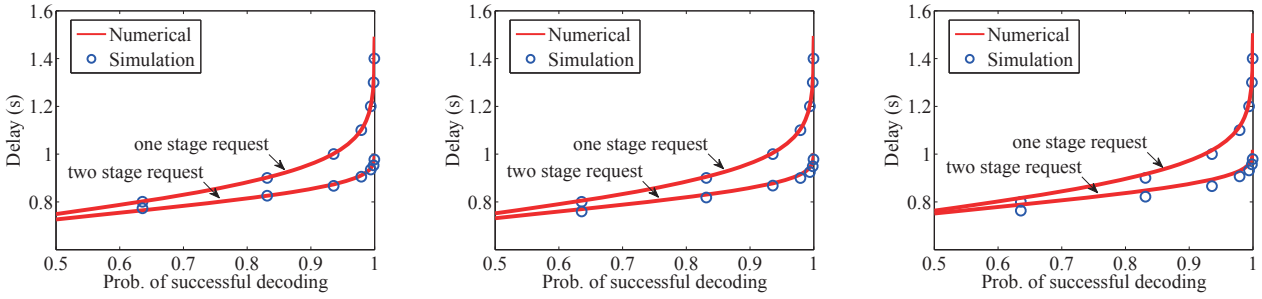
References

[1] Amazon S3. http://aws.amazon.com/s3/.



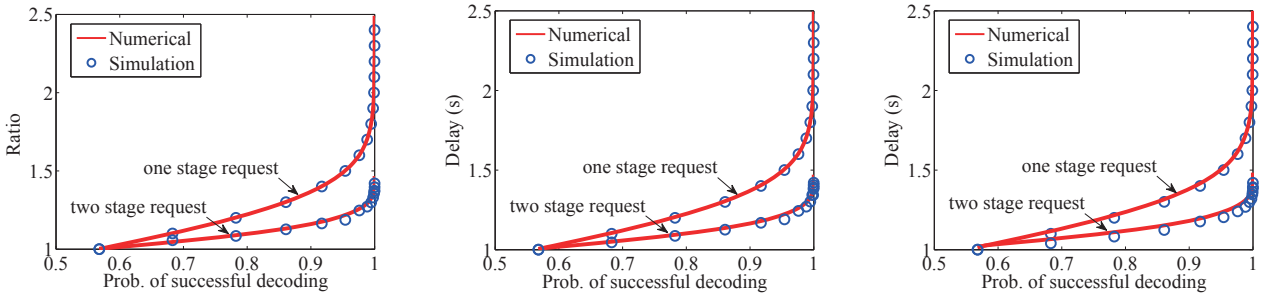
(a) $\lambda = 10, \theta = 25$ (b) $\lambda = 30, \theta = 50$ (c) $\lambda = 60, \theta = 50$

Fig. 9. Ratio of number of requested LT packets in first stage when $k = 100$.



(a) $\lambda = 10, \theta = 25$ (b) $\lambda = 30, \theta = 50$ (c) $\lambda = 60, \theta = 50$

Fig. 10. Tradeoff between decoding probability and delay when $k = 50$.



(a) $\lambda = 10, \theta = 25$ (b) $\lambda = 30, \theta = 50$ (c) $\lambda = 60, \theta = 50$

Fig. 11. Tradeoff between decoding probability and delay when $k = 100$.

[2] EMC Atmos Online Storage. http://www.atmosonline.com/.

[3] J. Hafner, "Weaver codes: highly fault tolerant erasure codes for storage systems," in *Proc. the 4th conference on USENIX Conference on File and Storage Technologies*. USENIX Association, 2005, pp. 16–16.

[4] J. Plank and L. Xu, "Optimizing cauchy Reed-Solomon codes for fault-tolerant network storage applications," in *IEEE Int. Symp. Network Computing and Applications*. IEEE, 2006, pp. 173–180.

[5] J. L. Cheng Huang, Minghua Chen, "Pyramid codes: Flexible schemes to trade space for access efficiency in reliable data storage systems," in *IEEE Int. Symp. Network Computing and Applications*. IEEE, 2007, pp. 79–86.

[6] F. Oggier and A. Datta, "Self-repairing homomorphic codes for distributed storage systems," in *Proc. IEEE INFOCOM*. IEEE, 2010, pp. 1215–1223.

[7] M. Sardari, R. Restrepo, F. Fekri, and E. Soljanin, "Memory allocation in distributed storage networks," in *Proc. Int. Symp. Information Theory ISIT*. IEEE, 2010, pp. 1958–1962.

[8] D. Leong, A. Dimakis, and T. Ho, "Distributed storage allocation for high reliability," in *IEEE Int. Conference on Communications (ICC)*. IEEE, 2010, pp. 1–6.

[9] A. Dimakis, P. Godfrey, Y. Wu, M. Wainwright, and K. Ramchandran, "Network coding for distributed storage systems," *Information Theory, IEEE Transactions on*, vol. 56, no. 9, pp. 4539–4551, 2010.

[10] D. Leong, A. Dimakis, and T. Ho, "Distributed storage allocations for optimal delay," in *Proc. Int. Symp. Information Theory ISIT*, 2011.

[11] M. Luby, "LT codes," in *Proc. 43rd Annual IEEE Symp. Foundations of Computer Science*, 2002, pp. 271–280.

[12] Z. Wilcox-O'Hearn and B. Warner, "Tahoe: the least-authority filesystem," in *Proceedings of the 4th ACM international workshop on Storage security and survivability*, 2008, pp. 21–26.

[13] A. Rényi, "On the theory of order statistics," *Acta Mathematica Hungarica*, vol. 4, pp. 191–231, 1953.

[14] R. Karp, M. Luby, and A. Shokrollahi, "Finite length analysis of LT codes," in *Proc. Int. Symp. Information Theory ISIT*, 2004, p. 39.

[15] E. Maneva and A. Shokrollahi, "New model for rigorous analysis of LT-codes," in *Proc. Int. Symp. Information Theory ISIT*, 2006, pp. 2677–2679.