Comparison of Data Replication Strategies for Peer-to-Peer Video Streaming

W. F. Poon
Department of Electronic and Information Engineering
The Hong Kong Polytechnic University
Hong Kong
enwil@eie.polyu.edu.hk

Jack Y. B. Lee and D. M. Chiu
Department of Information Engineering
The Chinese University of Hong Kong
Hong Kong
{yblee, dmchiu}@ie.cuhk.edu.hk

Abstract—**Current peer-to-peer (P2P) file-sharing systems are mostly optimized for file availability. This paper investigates P2P architecture for video streaming in general, and the performance impact of data redundancy schemes in particular. In particular, this work show that maximizing file availability is not the best strategy for video streaming as another constraint – peers' streaming bandwidth, comes into play. To address this limitation, a request-rate minimization policy is developed and evaluated using simulation. The resultant optimized replication strategy is then compared to data redundancy scheme based on erasure-correction coding. Simulation results show that with sufficient peer storage and a low erasure coding overhead, erasure-correction coding can achieve substantially better streaming performance than replication-based strategies.**

Keywords—**peer-to-peer, replication, video streaming**

I. INTRODUCTION

Providing on-demand video streaming services to large number of users in the Internet have long been a research challenge. In the literature many researchers have proposed various approaches to improve the scalability of media servers, such as the use of disk arrays [1], parallel servers [2], and distributed servers [3].

By contrast, researchers in recent years have turned their focus on another promising approach to solving this scalability problem – peer-to-peer (P2P) architectures [4-10]. Unlike client-server architectures, P2P systems take advantage of the rapid growth in computer processing, storage, and communication bandwidth in personal computers, and simply employ end-user machines to form a self-sufficient service network without the need for any dedicated servers or service operators.

There are many P2P file-sharing systems developed over the years. These can be classified into two categories: unstructured and structured [4]. Unstructured P2P systems, such as Napster, are centralized in that they have a central directory server for users to search and locate their desired files for download. Later generations of unstructured P2P systems such as Gnutella and KaZaA typically avoid such a central directory server by either using distributed search protocols or forming a two-level service network by dividing users into smaller groups, with a user host in each group elected as supernode to serve local search queries as well as to forward unresolved queries to other supernodes in the service network.

On the other hand, structured P2P systems, such as Chord [4], employed a distributed hash table (DHT) to organize peers into a highly-structured overlay network. A user can locate a data object through the DHT and this significantly improves search efficiency. However, if user machines are highly unreliable or join/leave the system frequently, such a structured approach may introduce more management overheads.

Despite the above-mentioned differences, current P2P systems are mostly designed for file sharing through download. Therefore the design goal is often to maximize file availability or reliability of the system [6-7]. While a small number of studies have investigated P2P architectures for video streaming, their focuses are primarily on transmission schedules [8] and fault tolerance mechanisms [9].

In this work we investigate a key issue in P2P video streaming systems – distributed data storage and its impact on video streaming performance. In particular, we develop a system model to analyze the streaming performance of replication-based P2P systems, incorporating the effect of data replication and placement policies.

We first briefly review some previous related work in Section 2, and then present the replication model and the erasure-correction-coding model in Section 3 and 4 respectively. These two models are then compared in Section 5 using simulation and finally we summarize the study in Section 6.

II. BACKGROUND AND RELATED WORK

Unlike dedicated media servers, end-user hosts by comparison will likely have smaller capacity (storage and bandwidth) and will be far more unreliable. Therefore an essential requirement in any P2P systems is data redundancy.

Data redundancy is typically implemented in one of two approaches, namely replication and

This work was funded in part by Earmarked Grant CUHK4211/03E from the HKSAR Research Grant Council, and the UGC Area of Excellence in Information Technology Scheme (AoE/E-01/99).

erasure-correction coding. In replication approaches [5, 7] multiple copies of the same file are distributed to different peers for storage. Thus a user can download the file as long as at least one copy of the file is available. By contrast, in erasure-correction coding approaches [8, 10] a file is first encoded using an erasure-correction code such as the Reed-Solomon Erasure Correction Code [11], and then divided into smaller pieces for storage in many different peers. As long as a certain minimum number of peers (and the coded file fragments) are available, the user can then download the fragments to recompute the original file data. Lin *et al*. [12] had analyzed and compared the data availability of these two data redundancy approaches and concluded that erasure-correction coding works better than replication when the peer availability is high, and vice versa.

However, these previous studies are all focused on file sharing using the download model, with file availability being the primary performance metric. In video streaming however, just downloading all the data are not sufficient – the data have to be received *in time* or else playback will not be continuous. This paper addresses this issue by developing a system framework for P2P video streaming systems that incorporated the effect of data replication and placement policies. Instead of file availability, the framework derives the probability of continuous playback as the performance metric. We first present and develop the frameworks for replication and erasure-correction coding in Section 3 and 4, and then compare them using numerical results in Section 5.

III. REPLICATION

In this section, we present and develop performance model for a replication-based P2P streaming system. The goal is not to provide a detailed system design but to develop a general system model that captures the essential properties of replication-based systems.

A. System Model

In a P2P system the availability of a peer primarily depends on how often the peer is on-line versus off-line. Host failure will also affect peer availability but the extent is overshadowed by the former factor. For simplicity we characterize the overall availability of a peer by a parameter β, known as the peer availability (or up time probability). Define T_{up} and T_{down} as the mean up time and mean down time duration respectively. Then the peer availability is equal to

$$\beta = \frac{T_{up}}{T_{up} + T_{down}}. \quad (1)$$

Note that peer availability is not the same as file availability. The latter also depends on the number of copies in the system (Section 3.2), placement of the files (Section 3.3), as well as the algorithm for peer selection during (Section 3.4). We first define the notations and present the details in the subsequent sections.

We consider a P2P service network of G peers, in which M ($G >> M$) of them are *serving peers* – peers that contribute storage and bandwidth to the service network, and the rest (G–M) peers are *free riders* – peers that do not contribute any storage and bandwidth to the service network. We denote the set of M serving peers by $P = \{p_1, p_2, \ldots, p_M\}$ and the set of J video objects by $V = \{v_1, v_2, \ldots, v_J\}$. The video v_j is replicated into s_j ($1 \le s_j \le M$) copies and we call the set $S = \{s_1, s_2, \ldots, s_J\}$ the *replication profile*.

Let b_j be the size of v_j in bytes. Assume each serving peer shares N_i bytes of storage, then a valid replication profile cannot exceed the total storage capacity of the system Λ:

$$\sum_{j=1}^{J} b_j s_j \le \Lambda = \sum_{i=1}^{M} N_i \quad (2)$$

The video copies are then distributed to s_j distinct peers in the set **P**. These serving peers provide all the storage and streaming bandwidth to serve all the peers (including serving peers and free riders) in the system.

We assume video streaming requests form a Poisson process with mean arrival rate λ. The video popularities $\{q_j \mid j=0,1,\ldots,(J\text{-}1)\}$ follows the Zipf distribution [13]. Thus the arrival rate for video v_j is given by $\lambda_j = \lambda q_j$. For example, if all the serving peers are "up", the average request rate for v_j, which there are s_j copies, observed by a serving peer storing a replica of v_j is given by

$$w_j = \frac{\lambda q_j}{s_j}. \quad (3)$$

B. Replication Profile

With this replication-based system model, the first problem is determining the replication profile **S**, i.e., the number of replicas for each video. In *file-sharing* systems, the typical goal is to maximize file availability or hit rate (probability that a requested file is available). If the peer availability and video popularity are known *a priori*, then this problem can be formulated into a constrained maximization problem [14]:

$$\text{Maximize } P_{hit} = 1 - \sum_{j=1}^{J} q_j (1-\beta)^{s_j} \quad (4)$$

Subject to (2) and $1 \le s_j \le M$.

This optimization problem (henceforth called OP1) can be solved using dynamic programming [15] to obtain the optimal replication profile.

Nevertheless, this optimization model does not capture the requirement of video streaming. In particular, it does not account for the serving peers' streaming bandwidth constraint. For example, a video request will still fail if all the serving peers' streaming bandwidth is fully utilized, although the video file is available in the file availability sense.

To account for this streaming bandwidth constraint, we therefore need to revise both the objective function as well as constraints in the optimization model. Specifically, instead of maximizing file availability, we choose to minimize the average request rate received by the serving peers. Given that the streaming bandwidth is fixed and cannot be changed dynamically, minimizing the average request rate will also minimize the request blocking probability. This new optimization model, henceforth called OP2, is given by

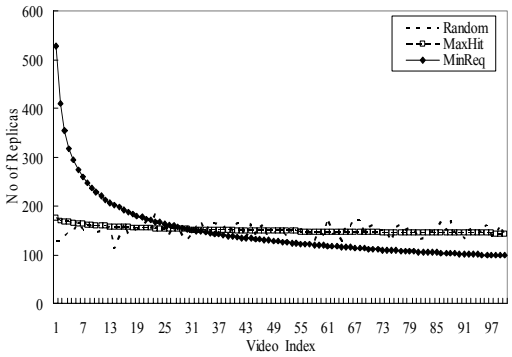

Figure 1. Comparison of replication profiles

$$\text{Minimize} \quad P_{req} = \sum_{j=1}^{J} \frac{\lambda q_j}{\beta s_j} \tag{5}$$

Subject to (2) and $1 \le s_j \le M$.

Again this optimization problem can be solved using techniques such as dynamic programming to obtain the optimal replication profile.

Fig. 1 compares the replication profiles generated from randomization, MaxHit (OP1 in Eq. (4)), and MinReq (OP2 in Eq. (5)). The result for the randomized replication profile is expected but the other two policies generated very different replication profiles. In particular, the MaxHit policy is insensitive to the video popularity skew and generated about the same number of copies for all the videos. This is counter-intuitive because one will expect to allocate more replicas for popular videos. The reason for this result is that the MaxHit policy does not account for peers' bandwidth constraint. In other words, as long as at least one peer storing the video is online, the video is considered to be available, regardless of whether this single peer will have sufficient bandwidth capacity to serve all the requests.

This is also why the MaxHit policy widely used in P2P file-sharing systems is not suitable for P2P video streaming. This limitation is addressed by the MinReq policy, which now properly allocates more replicas to popular videos to balance the load across all the serving peers.

C. Placement Policy

Knowing the replication profile, we need to determine which serving peers to store which video files – the placement policy. At first it may appear that it does not matter which videos are stored in each peer as long as the serving peers divide all the video files equally. However, this is true only if the videos all have the same popularity or when $s_j = M \ \forall \ j$, i.e., there is sufficient storage to replace all videos over all peers. For example, if the popular videos are all concentrated in the same set of serving peers, then these serving peers will experience heavier video request load than the rest of the serving peers storing unpopular videos, leading to load imbalance among the serving peers.

To balance the load across all the serving peers, we employ a well-known placement policy called the Smallest Load First (SLF) proposed by Zhou and Xu [13]. In the iterative SLF algorithm, all the video replicas are first sorted in non-increasing order according to w_j computed from (2). In each iteration the system selects M video replicas with the largest w_j and then distributes these M replicas to M different peers according to two rules: (a) the replica with the largest w_j is stored in the peer with the smallest load - computed from $l_i^k = \sum_{v_j \in Z_i^k} w_j$ where Z_i^k is the set of video replicas stored in peer i after the k^{th} iteration.; and (b) a peer cannot store more than one replica of the same video.

D. Peer Selection Scheme

After video replicas are distributed across the serving peers, we need a peer selection scheme to select one of the available serving peers during the admission of a new streaming request. Again the goal is to balance the load across the available serving peers.

We assume that through some distributed directory service (e.g., distributed search or DHT) a requesting peer can obtain the list of available serving peers storing the replica of the requested video. Armed with this list, one simple peer selection scheme will be to randomly choose one of the available peers as the sender for the new streaming session. Taking this one step further we can also attempt to explicitly balance the serving peers' load by using algorithms such as the Least Load First (LLF) algorithm proposed by Li *et al*. [16]. To apply the LLF algorithm, the requesting peer will request the serving peers' available transmission bandwidth during admission, and choose the one with the maximum available bandwidth as the sender.

Nevertheless, even a serving peer can be identified for the new streaming session, it may still fail during the session. In this case, the requesting peer has no choice but to find another serving peer as replacement, using the same admission algorithm as described earlier. If no serving peer is available, then the playback will have to be stopped and the streaming session considered to have failed. In our simulation in Section 5 we use the probability of successful playback, defined as the proportion of requests that can completely playback the whole video, as the performance metric.

IV. Erasure-Correction Coding

In addition to replication, another way to implement data redundancy is to employ erasure-correction coding such as the Reed-Solomon Erasure Correcting (RSE) Code [11] on the video data [19]. Specifically, we first divide the video into many small fragments (say a few kilobytes each). Using erasure-correction code we then generate h redundant fragments from every (M-h) data fragments. Each of the M fragments in each group will then be distributed to M distinct serving peers. The key is that a requesting peer can reconstruct the original (M-h) data fragments as long as *any* of the M fragments in a group are received, thus enabling the receiver to recover from up to h node failures [19].

Let h_j be the redundancy level for v_j. Then the extra storage needed for the redundant data of v_j is

$$e_j = \frac{M}{(M - h_j)}. \tag{6}$$

To start a new streaming session the requesting peer again will use a distributed directory service to locate the M serving peers, and then randomly select (M-h) of the serving peers for the new streaming session. Unlike in replication, a streaming session under erasure-correction coding will comprise (M-h) serving peers sending data simultaneously to the requesting peer. Given a video bit-rate of C bps, the (M-h) serving peers will then split the video bit-rate equally, resulting in a per-peer transmission rate of

$$B = \frac{C}{M - h}. \quad (7)$$

V. Performance Comparisons

In this section, we use simulation to evaluate and compare the streaming performance of replication-based and erasure-correction-coding-based data redundancy schemes. We assume that request arrivals form a Poisson process and the video popularity follows the Zipf's distribution with a skewness of θ=0.271 [13]. The peer availability is set to 0.1 according to the measurement study by Saroiu *et al*. [18], implemented using an exponentially distributed up and down time of mean T_{up}=60 and T_{down}=540 minutes respectively. The system stores a total of 100 videos with the same length of 120 minutes. The uplink bandwidth of the serving peers is twice that of the video bit rate and the downlink bandwidth of the peers (including serving peers and free riders) is unbound. Performance is measured by the probability of successful playback as defined in Section 3.4.

We first compare the performance impact of different replication strategies in Fig. 2 for request arrival rate from 0.01 to 0.1. There are 1,500 serving peers, each having the capacity to store 10 videos. Three replication strategies are simulated, namely "Random" (random replication profile, random placement policy, and random peer selection scheme), "MinReq_SLF_LLF" and "MaxHit_SLF_LLF", where the replication profile is either generated by maximizing the hit rate ("MaxHit") or minimizing the request rate ("MinReq").

The results show that if we employ the traditional "MaxHit" replication profile, then even optimizing the placement policy (using SLF) and the peer selection scheme (LLF) will not give any significant improvement over the randomized algorithm.

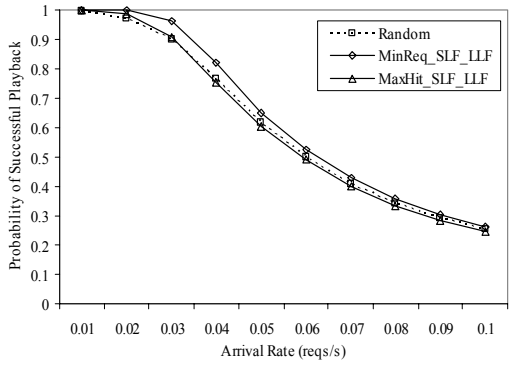

Figure 2. Comparison of replication policies

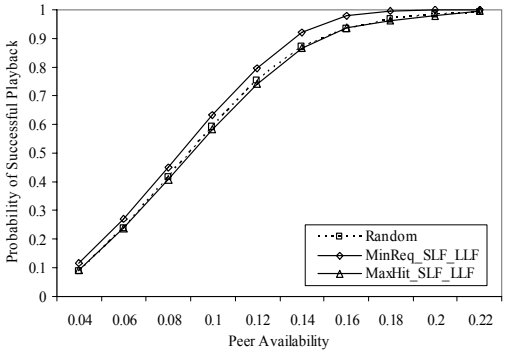

Figure 3. Performance impact of peer availability

This is because the solution space of placement and peer selection optimization is largely determined by the replication profile, and as Fig. 1 shows, it is similar to the randomized replication profile. On the other hand, the "MinReq" replication policy does allow better optimization of placement and peer selection, resulting in more than 6% improvement in the successful playback probability at an arrival rate of 0.03.

Next we investigate in Fig. 3 the impact of peer availability on streaming performance, again using replication as the data redundancy scheme. We plot the probability of successful playback for peer availability ranging from 0.04 to 0.22 [18]. There are 1,200 serving peers and the mean request arrival rate is 0.04 reqs/s. We observe that the successful playback probability drops progressively as the peer availability decreases. For example, at a peer availability of 0.1, the successful playback probability is 0.63 under "MinReq_SLF_LLF".

In the next set of simulations, we compare the streaming performance of replication against erasure-correction coding. Fig. 4 and 5 compare the successful playback probability versus number of serving nodes for two peer storage capacities. For the case of erasure-correction coding, we introduce an overhead parameter γ, defined as the proportion of peer's uplink bandwidth, to account for the additional overheads (e.g., control traffic, directory service, system management, etc.) consumed in managing and maintaining the more complex erasure-correction-coded data storage.

From Fig. 4, we observe that erasure-correction coding generally outperforms replication, provided that the overhead is less than 0.2. On the other hand, if we reduce the peer storage capacity from 10 videos to 2 videos, erasure-correction coding will outperform replication by a significantly larger margin as shown in Fig. 5. This is because when peer storage is abundant, there are large numbers of replica for each video. Consequently the placement and peer selection algorithm can distribute the load across more serving peers, thus resulting in better load balance. By contrast, if the peer storage is very limited, then any peer storing a replica of a popular video will likely experience a larger proportion of the streaming requests. Without many replicas to choose from, some peers (those storing the popular videos) will be more heavily loaded than others (those storing the less popular videos).

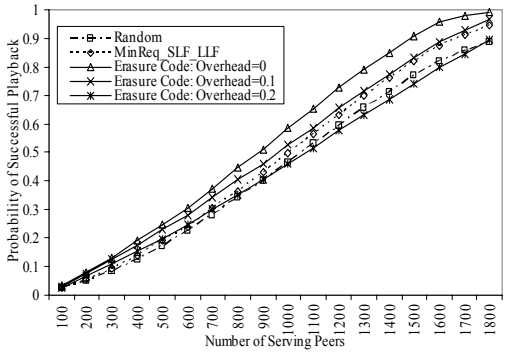

Figure 4. Comparison of replication and erasure-correction coding (peer storage = 10)

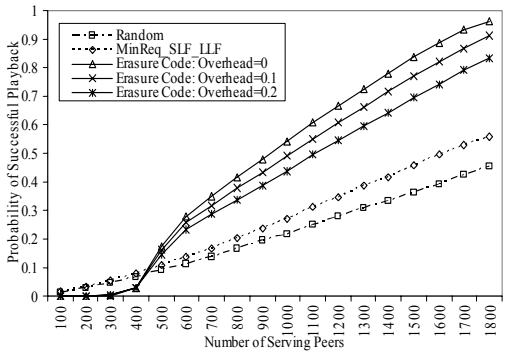

Figure 5. Comparison of replication and erasure-correction coding (peer storage = 2)

Nevertheless, when the number of peers drops below 500, the streaming performance of erasure-correction-coding collapses to below that of replication. This is because at these extreme scenarios there are simply too few peers available for successful erasure-correction computation, which requires at least $(M\text{-}h_j)$ out of the M peers to be available. Replication, on the other hand, will still work at these extreme scenarios as long as at least one peer is available.

IV. CONCLUSIONS

In this work we developed compatible system models for P2P streaming systems built upon replication and erasure-correction coding data redundancy schemes. Our results revealed that the current availability-maximizing policy adopted in file-sharing systems is not suitable for streaming applications as the peers' bandwidth constraints are not accounted for. Erasure-correction coding, on the other hand, can provide additional gain in performance, but only if the number of peers is not too small. Hence one still needs to consider the particular system configuration in order to choose the better-performing data redundancy scheme. We are currently developing analytical models to capture the essential properties of the system so that one does not need to resort to time-consuming simulations to determine the appropriate data redundancy scheme.

REFERENCES

[1] F. A. Tobagi, J. Pang, R. Baird, and M. Gang, "Streaming RAID – a Disk Array Management System for Video Files," *Proc. ACM Conf. Multimedia '93*, 1993.

[2] Y. B. Lee and P. C. Wong, "Performance Analysis of a Pullbased Parallel Video Server," *IEEE Trans. on Parallel and Distributed Systems*, 11(12): 1217–1231, 2000.

[3] D. N. Serpanos and A. Bouloutas, "Centralized versus distributed multimedia servers," *IEEE Trans. on Circuit Systems and Video Technology*, vol. 10, no. 8, pp. 1438-1449, Dec. 2000.

[4] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," *SIGCOMM 2001*, San Diego, CA.

[5] S. Iyer, A. Rowstron and P. Druschel, "SQUIRREL: A Decentralized, Peer-to-Peer Web Cache," *PODC*, 2002.

[6] F. M. Cuenca-Acuna, C. Peery, R. P. Martin and T. D. Nguyen, "PlanetP: Using Gossiping to Build Content Addressable Peer-to-Peer Information Sharing Communities," *Proc. IEEE Int'n Symposium on HPDC*, 2003.

[7] A. Adya, William J. Bolosky, M. Castro, G. Cermak, R. Chaiken, John R. Douceur, J. Howell, Jacob R. Lorch, M. Theimer, and Roger P. Wattenhofer, "FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment," *OSDI 2002*.

[8] C. Y. Chan, Jack Y. B. Lee, and M. Hamdi, "Gradient-Descent Scheduler - Network-Aware Transmission Scheduler for Server-less Video Streaming Systems," *ICC 2005*, 2005.

[9] T. Do, K. A. Hua, and M. Tantaoui, "P2VoD: Providing Fault Tolerant Video-on-Demand Streaming in Peer-to-Peer Environment," *ICC 2004*, 2004.

[10] S. Rhea, P. Eaton, D. Geels, H. Weatherspoon, B. Zhao and J. Kubiatowicz, "Pond: the OceanStore Prototype," *FAST 2003*.

[11] J. S. Plank, "A Tutorial on Reed-Solomon Coding for Fault-Tolerance in RAID-like Systems," *Software Practice and Experience*, vol. 27(9), Sep. 1997, pp.995-1012.

[12] W. K. Lin, D. M. Chiu, and Jack Y. B. Lee, "Erasure Code Replication Revisited," *Proc. 4th Int'n Conf. on Peer-to-Peer Computing*, 2004.

[13] X. Zhou and C. Z. Xu, "Optimal Video Replication and Placement on a Cluster of Video-on-Demand Servers," *Proc. IEEE 31st ICPP*, pp 547-555, 2002.

[14] J. Kangasharju, K. W. Ross, D. Turner, "Adaptive Content Management in Structured P2P Communities," http://cis.poly.edu/~ross/p2pTheory/ P2Preading.htm.

[15] Ibaraki T., Katoh N., Resource Allocation Problems – Algorithmic Approaches, The MIT Press, Cambridge, MA.

[16] T. S. Li, C. M. Chang, J. M. Ho and M. T. Ko, "Dynamic load adjustment on distributed video server system," *Proc. SPIE*, vol. 3229, p. 103-112, 1997.

[17] H. Weatherspooon and J. D. Kubiatowicz, "Erasure Coding vs Replication: A Quantitative Comparison," *IPTPS 2002*.

[18] S. Saroiu, P. K. Gummadi, and S. D. Gribble, "A Measurement Study of Peer-to-Peer File Sharing Systems," *Proc. MMCN*, 2002.

[19] Jack Y. B. Lee and W. T. Leung, "Design and Analysis of a Fault-Tolerant Mechanism for a Server-Less Video-On-Demand System," *Proc. Parallel and Distributed Systems*, 2002.