A Framework for Lazy Replication in P2P VoD*

Bin Cheng
Huazhong University of Science and Technology
showersky@hust.edu.cn

Lex Stein
Microsoft Research Asia
castein@microsoft.com

Hai Jin
Huazhong University of Science and Technology
hjin@hust.edu.cn

Zheng Zhang
Microsoft Research Asia
zzhang@microsoft.com

ABSTRACT

Video-on-Demand (VoD) is a compelling application, but costly due to the load it places on servers. Peer-to-peer (P2P) techniques hold the potential to reduce centralized costs by sharing data between peers. There are many difficult design issues associated with P2P for VoD. Viewing the problem as designing a large distributed cache, many of the issues can be expressed in terms of caching algorithms.

In an earlier paper [6], we studied the performance of GridCast, a P2P VoD system deployed on CERNET. From system traces, we found that departure misses are the major cause of server load. Motivated by this finding, this paper examines how to use replication to decrease departure misses and thereby further reduce server load.

This paper proposes and evaluates a framework for lazy replication. Lazy replication postpones replication, trying to make efficient use of bandwidth. In our framework, two predictors are plugged in to create the working replication algorithm. Lazy replication with several predictors is compared with a naïve eager replication algorithm. We find that lazy replication is more efficient than eager replication, even when using two simple predictors. With these two simple predictors, lazy replication can decrease server load by 15% from multivideo caching with only a minor increase in network traffic.

1. INTRODUCTION

Video-on-Demand (VoD) is increasingly popular with Internet users. However, VoD is costly due to the load it places on video servers. Peer-to-peer (P2P) techniques share between peers to reduce server load. There are a number of difficult design issues associated with P2P for VoD.

For the purpose of systematically evaluating the performance of P2P VoD and exploring the room for further optimizations, we have built and deployed GridCast [5] [6]. GridCast is the first internet VoD system to provide a full set of VCR operations, such as pause, forward, random seek, while using P2P to reduce server load and improve user experience. It has been live on China's national CERNET[1] since May of 2006. In peak months, GridCast has served videos to 23,000 users.

In GridCast, each peer caches fetched chunks to increase the content for sharing. The caching is purely passive and demand-driven. A peer will only issue a fetch for the purpose of satisfying its current viewing needs. Observations of GridCast show that caching can decrease server load by 36% from client-server [6]. However, that still leaves a remaining 64%. Our past work found that much of this remaining server load was for chunks that were hosted on one or more peers at some time in the past, but all such peers have since departed. These chunk fetches, which we term *departure misses* are responsible for 43% of the remaining server load. Departure misses are a major issue.

Reducing these departure misses is the aim of this paper. Replication can help caching facilitate sharing between peers. With replication, a peer sends chunks to other peers in a proactive way, not simply to satisfy immediate viewing needs. This means a peer may cache chunks that it has not watched or does not even plan to watch. By using more peer disk and network resources, replication can increase the lifetimes of chunks in the P2P cache, and thereby reduce departure misses. Replication also has its costs. Caches and bandwidth are finite and replication increases contention on them, possibly increasing chunk misses for these reasons. Finding balance on the tradeoff between costs and benefits is a difficult design issue.

With this question in mind, this paper introduces an algorithmic framework for lazy replication. Chunk request and peer departure predictors are plugged into the framework to construct a replication algorithm. This paper evaluates simple predictors and compares them against a naïve eager replication algorithm. Through trace-driven simulation we examine their effectiveness. Our results include several important insights. First, lazy replication can further decrease server load by 15% from caching, using only simple predictors. Second, lazy replication obtains this improvement at a much lower cost than eager replication. Finally, in the

*This work is supported by National Natural Science Foundation of China (NSFC) grants 60703050, 60433040 and 60731160630, Research Fund for the Doctoral Program of Higher Education grant 20050487040, Wuhan Chengguang Plan grant 200850731350, as well as grants from Microsoft Research Asia.

NOSSDAV '08 Braunschweig,Germany

[1]CERNET stands for China Educational and Research Network. As of January 2006, CERNET connects about 1500 institutions and about 20 million end users.

same algorithmic framework, simple predictors achieve performance similar to predictors with knowledge of one hour into the future.

This paper is organized as follows. Section 2 starts with analysis of the GridCast logs, then discusses the motivation and challenges for replication in P2P VoD. Section 3 presents the framework for lazy replication and its two building blocks; the peer departure predictor and the chunk request predictor. Section 4 evaluates lazy replication and compares it to eager replication. Section 5 describes the related work, then Section 6 concludes.

2. MOTIVATION

This section first describes an overview of the GridCast system. Based on the trace collected from GridCast, we then analyze why source fetches (misses) cannot be served by a peer and discuss the challenges and opportunities for replication.

2.1 Overview of GridCast

A GridCast system comprises a track server (tracker), one or more video source servers (sources), peers, and a web portal. The tracker is a well-known rendezvous for joining peers. It maintains a membership list of all joined peers to facilitate data sharing between peers. The sources store a persistent and complete copy of every video. Videos are partitioned into chunks, each with a fixed playing time of 1s. Peers fetch chunks from sources or peers and cache them in local memory and disk, evicting by LRU. Peers refresh their playhead information every 30s, and synchronize with the tracker every five minutes or on a user seek to obtain more candidate peers for sharing.

GridCast hosts about 2,000 video files, uploaded and categorized by the system administrators. The videos are movies and television shows and are classified into nine categories, ranging from action movies to variety shows. The time length of the video files ranges from five minutes to two hours, with an average length of 48 minutes. Movies make up 46.5% of the video files and TV shows the other 53.5%. Videos are encoded from 400 Kbps to 800 Kbps, with an average bitrate of 610Kbps.

In GridCast, peers cache all their recently viewed chunks locally in memory and the file system. Chunks are evicted by LRU when the cache reaches a maximum of 1GB. This caching algorithm is called multivideo caching (MVC), in contrast to an earlier caching algorithm that only cached chunks from the video being played (single video caching, or SVC).

The traces used in this paper were taken from August 3, 2007 to August 30, 2007. This was after the deployment of multivideo caching and is therefore refered to as the MVC trace.

The trace logs system events and latency observations at a fine level of detail. It includes all user joins and leaves, all VCR operations, and all chunk requests and transfers. This detailed logging enables us to perform deep analysis. Table 1 summarizes the statistics of the MVC trace.

2.2 Analysis of Misses

To understand the causes of source server load, the traces are analyzed to classify each source fetch, determining why it was not served by any peer. This requires detailed knowledge of global system state. We reconstruct this knowledge by rerunning the log events then investigating global peer state at the time of each source fetch. Each source fetch is classified into one of the following five categories.

Statistic	MVC trace
Time length	28 days
Time period	08/03/07 - 08/30/07
Number of unique users	14,000
Max number of active users	300
Number of videos	2,000
Number of video views	84,000
Number of user sessions	62,000
Total data served from sources	14,200 GB
Total data served from peers	7,900 GB
Playing time	35,000 hours
Users behind a NAT	23.7%

Table 1: Statistics during deployment
Dates are in MM/DD/YY format. MVC denotes multiple video caching.

Insufficient bandwidth. Some peers in the neighborhood cache the chunk, but there is no available bandwidth to any of them.

Connection issue. One or more active peers cache the chunk, but there are no available connections to any of them. There are two reasons why a peer cannot connect to another. First, the other peer is behind a NAT. Second, the other peer has reached its neighbor connection limit.

Peer eviction. No active peers cache the chunk. An active peer once cached the chunk, but evicted it. A larger cache size can further decrease these misses.

Peer departure. No online peers cache the chunk. At one time, one or more peers cached the chunk, but all such peers have since departed.

New content. The content has never been fetched by any peer. It is completely new. There is no way to eliminate these misses.

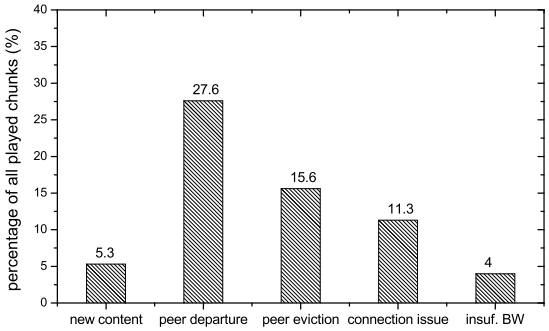

Figure 1: Breakdown of misses by cause

For the MVC trace, Figure 1 breaks down the misses by cause. Note that server load is decreased by 36.2% from a client-server architecture (with no caching or prefetching). The source fetches account for 63.8% of all played chunks, including *new content* (5.3%), *peer departure* (27.6%), *peer eviction* (15.6%), *connection issue* (11.3%), and *insufficient bandwidth* (4.0%).

Peer departure is the dominant cause, responsible for about 43% of the total misses. Caching cannot reduce departure misses, even if a peer were to have an infinite cache size. Replication can increase the lifetime of chunks to eliminate some of these departure misses. However, the net effect of replication is an open question. This paper focuses on how

to use replication to eliminate departure misses, within prior cache size and bandwidth capacity limits.

2.3 Challenges and Opportunities

VoD has different properties than other network media applications. In VoD, the average session time is shorter than in file downloading. For example, in Maze a popular file sharing system also deplyed on CERNET, the mean user session time is 4.5 hours [9]. In GridCast, the mean user session time is 1.6 hours and 40% of user sessions are less than 10 minutes. In addition, users join and leave the system frequently in GridCast. The average probability of peer departure is 50% for each hour. These properties make VoD particularly challenging.

Replication needs to use additional peer network bandwidth and disk space. To determine if there are resources available, we compute the existing resource utilization.

For each peer, its peak upload and download bandwidth use is computed by scanning the log events, and computing the total upload and download across every 10s interval, then selecting the maximum values across all intervals. Using this technique, across all peers, the average peak download is 3.8Mbps. This is higher than the computed average peak upload of 2.2Mbps. These peaks are conservative estimates for bandwidth limits. With these conservative estimates, unused download and upload capacity are 72% and 81%, respectively.

Replication also uses additional disk resources. The log trace lacks information on peers' physical disk sizes. However, across all peers, the average disk cache utilization is 63%. The disk cache is necessarily smaller than the physical disk, indicating that there is more than 37% available disk.

Taken together, these results suggest there are peer disk and network resources available for replication.

3. REPLICATION

The purpose of replication is to decrease source fetches by proactively changing the chunk distribution over peers. Replication has two benefits. First, it reduces departure misses. Second, it might reduce eviction misses if chunks are replicated to a peer with free cache space. Replication has three costs. First, it uses more peer network and disk resources. Second, it increases connection misses and insufficient bandwidth misses. Third, it can increase cache pressure and eviction misses. A good replication algorithm can maximize the benefits and minimize the costs.

3.1 Replication Algorithm

A replication algorithm must answer three key questions; what to replicate, where to replicate to, and when to replicate?

3.1.1 Eager Replication

In eager replication, when a peer fetches a chunk from a source, it enqueues a command to replicate the chunk. Whenever it has idle upload, the peer processes the command queue. All chunks are replicated with a fixed *redundancy factor* (K). To process the command from the front of the queue, the algorithm greedily replicates to the K longest-lived partners first, subject to their download and cache usage constraints. The choice of pushing to the longer-lived peers is based on the observation that peers that have been around longer tend to stay around longer.

3.1.2 Lazy Replication

Lazy replication is an algorithmic template parameterized by two subalgorithms; a peer departure predictor and a chunk request predictor. The peer departure predictor predicts if a peer will depart in the next time interval. The chunk request predictor predicts the future popularity of a chunk, in terms of plays.

The replication algorithm decides which chunks to replicate, to which peers, and when, given uncertainty about the future. It is impossible to know exactly when peers will depart and what chunks will be requested. The predictors approximate this knowledge. Lazy replication uses a peer departure predictor to choose when to replicate (just before a peer leaves) and to whom (peers that are predicted to be most unlikely to leave) and uses a chunk request predictor to choose what to replicate (the chunks that will be the most popular in the coming sessions).

Lazy replication uses the same greedy approach as eager replication; replicate the chunk with the highest popularity to the peer with the lowest probability of departing. The two differ in when a chunk is replicated. Lazy replication uses the peer departure predictor to decide both when to replicate and to where.

Peers do run other applications and replication cannot monopolize all upload all the time. A lazy factor, α, controls the upload used for replication. If a peer does not predict departure, it replicates less. If a peer predicts that its departure is imminent, it replicates using as much upload as it can. Eager replication is more aggressive than lazy replication because it immediately replicates whenever a peer fetches a chunk.

A peer makes the decisions for lazy replication locally as follows. The peer manages a neighborhood set and two lists, a list of candidate chunks and a list of target peers. The neighborhood is a set of online peers chosen for downloading and uploading. The list of candidate chunks contains all locally cached chunks that do not have a copy on any other peer in the neighborhood. The candidate chunks are sorted by their predicted future popularity. The list of target peers contains all peers in the local neighborhood, and is sorted by the peers' online time. To replicate, the peer selects the chunk with the highest popularity from the candidate list, and sends it to a target peer that is selected from the sorted target peer list. We develop and examine several different ways for target peer selection in Section 4.2.1 later. The peer repeats selecting chunks until exhausting its upload for the interval. Then, the peer resets its upload and download capacity for the next interval and repeats. Figure 2 shows pseudocode for lazy replication.

3.2 Simple Predictors

As described above, lazy replication takes two predictors. This section discusses the motivation for the implementation of two simple predictors. These two predictors are plugged into lazy replication to construct the ***lazy-simple*** replication algorithm.

It is likely that more accurate predictors can improve the efficiency of replication. To some extent, the performance of a lazy replication algorithm will depend on the accuracy of its predictors. To evaluate the sensitivity of performance to predictor accuracy, we propose two predictors with future knowledge, called oracle predictors. Though called oracle, these predictors do not have perfect knowledge of the fu-

```
Δt: replication period, (Δt = 10seconds)
α: lazy factor
K: redundancy factor
void do_replication(){
  //(1) run predictors to get predicted value
  departure_predictor();
  chunk_request_predictor();
  //(2) pick candidate chunks and target peers
  if (bLeave == true) up_capacity = up_capacity×α;
  select_candiate_chunks();
  select_target_peers();
  //(3) try to do replication
  while(true){
    pick the first chunk i from the candidate chunk list
    if (the size of chunk i > up_capacity) break;
    pick K target peers from the target peer list
    replicate chunk i to the selected K target peers;
    up_capacity = up_capacity - size of chunki × K;
  }
  //(4)reset the download and upload capacity for each peer
  up_capacity = max_up_bw×Δt;
  down_capacity = max_down_bw×Δt;
}
```

Figure 2: Lazy replication pseudocode.

ture. They only know with certainty what chunk requests and peer departures will happen in the next hour. Lazy replication parameterized by these two oracle predictors is called ***lazy-oracle***. Lazy-oracle is not necessarily optimal. First, the predictors know only one hour into the future, lacking complete future. Second, like eager replication, the lazy replication framework is a greedy algorithm and it is not known to be optimal.

Lazy-oracle and lazy-simple share the same algorithmic framework (Figure 2). But they plug different predictors into this framework. Section 4.2.2 compares these two algorithms. The next two subsections describe the predictors of lazy-simple.

3.2.1 Simple peer departure predictor

Past behavior can often be an accurate predictor of future behavior. The time a peer has been online might indicate its likelihood of departure. For example, a peer online for 5m may be more likely to leave in the next minute than a peer online for the past day. Figure 3 shows the PDF of user session durations. Of all user sessions, 50% spend less than 10m online. For sessions exceeding 10m, the percentage of user sessions with longer online time decreases sharply. The last bar includes all user sessions online for more than 2h.

This data shows that online time is a predictor of peer departure. The simple departure predictor is based on this observation. It uses only online time to predict the departure of a peer. A peer is predicted to leave if it has been online for fewer than ΔI minutes. Otherwise, it is predicted to stay in the system.

We evaluate the precision of this simple departure predictor. When the interval (ΔI) is 10 minutes, both precision and recall are 54%. When the interval is 60 minutes, both precision and recall increase to 78%. A larger interval has better precision, but that means replication starts earlier.

3.2.2 Simple chunk request predictor

Yu et al. [11] observe that the popularity of a video changes during a typical day and the change between two continuous short intervals is smooth. In this case, the popularity in the most recent interval is useful to predict the future requests. In addition, chunks requested recently are more likely to be

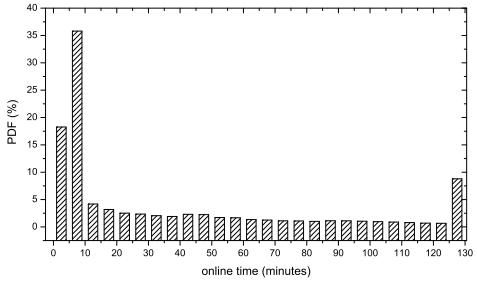

Figure 3: Online time.

requested earlier. Based on this observation, the predictor weights the contribution of recent requests to decay with time.

For a time of ΔM, a prediction interval of ΔI, time is split into n ($n = \frac{\Delta M}{\Delta I}$) ranges. The ranges $r_1, r_2, ..., r_n$ are sorted by decreasing time. The first range, r_1, is the most recent range and is given the greatest weight. Figure 4 shows how the simple chunk request predictor works.

```
ΔI: time interval for chunk request predictor
ΔM: time to look back
K: K=∑_{i=1}^{ΔM/ΔI} (1/i)
L[j]: the size of chunk set for peer j
r[n]: the number of requests in the time range r_n
R_i: the predicted number of request for chunk i
void chunk_request_predictor(){
  for(int i=0; i<L[j]; i++){
    R_i = (1/K) × ∑_{n=1}^{ΔM/ΔI} (r[n]/n)
  }
}
```

Figure 4: Chunk request predictor.

We measure the error of the chunk request predictor with different parameters. The error is calculated by the absolute difference between the real value and the predicted value. A chunk is overestimated if its predicted value is larger than its real value. Otherwise, it is underestimated. Table 2 presents the results. For a given interval, a longer history time is helpful to decrease the error of prediction. For example, when the interval is 60 minutes and the history time increases from 3 hours to 6 hours, the average error decreases from 0.80 to 0.62. That is because the predictor has more information to refine the predicted result. Note that nearly 98% of chunks are overestimated, given a interval and a history time. The result shows that the chunk request predictor turns to overestimate its predicted value. This leads to the fact that some chunks are selected to be replicated but in reality they have no more requests happened in the future.

history time (hours)	3		6	
interval (minutes)	10	60	10	60
average error (req/h)	1.68	0.80	1.15	0.62
overestimated (req/h) (98%)	1.62	0.81	1.09	0.62
underestimated (req/h) (2%)	4.94	0.93	5.14	0.89

Table 2: Evaluation of request predictor.

4. PERFORMANCE EVALUATION

This section first introduces the experimental environment and the metrics used to evaluate the performance of our algorithms. Then, lazy and eager replication are compared using trace-driven simulation.

4.1 Simulation Setup

The evaluation uses a two-week log, taken from a part of the MVC trace. The first week, from August 6, 2008 to August 12, 2008, is used to construct the peer departure and chunk request predictors. The second week, from August 13, 2008 to August 19, 2008, is used to evaluate how well replication works based on these predictors. Caching is implemented in the simulator, staying faithful to the salient features of the deployed algorithm. In the simulation, each peer has a fixed cache size, up to 1GB. The simulator ignores NATs, but considers connection limits. The redundancy factor (K) is set to 1 in all experiments. The interval time of the peer departure predictor and the chunk request predictor is 10 minutes and 1 hour respectively.

The simulation needs an estimate of each peer's upload and download capacity. To get these limits, it uses the estimation technique described in Section 2.2. As discussed in that section, this approach is conservative because many peers are short-lived and will not have the opportunity to realize their bandwidth peaks.

Replication decreases source server load by eliminating departure misses. The simulation measures the source server load by the total number of source fetches. The reduction of source server load represents the benefit achieved from replication. Replication also increases network traffic between peers. This is measured by the total number of additional chunks replicated between peers. The efficiency of replication is defined in terms of its marginal benefits over its marginal costs where the marginal benefits are the reduction in server load and the marginal costs are the additional induced network load. Together, the reduction in server load and marginal increase in network traffic are accounted in the evaluation of replication.

4.2 Evaluation Results

The experiments of this section first examine the performance of lazy-oracle with different configurations to find the best configuration. Under the best configuration, lazy-oracle, lazy-simple, and eager replication are all compared using the metrics of server load reduction and efficiency (described above in Section 4.1).

4.2.1 Examining Configurations

In the lazy replication framework, a replicating peer selects a target peer from its local target peer list. The choice of target can be done in a number of ways. For example, random or sequential. With random selection, the replicating peer replicates to a random target peer. With sequential selection, the replicating peer replicates to the target peer with the lowest predicted probability of leaving. These methods are compared using simulation.

The sequential and random methods of choosing the target are compared in Figure 5. The figure also includes results for a method called *file locality first*, described below. The x-axis labels represent several different things. The *replication* counts the total number of chunks replicated. The *SS load* counts the number of chunks fetched from the server. The following five labels categorize the misses (source fetches), similar to Figure 1.

Random selection has higher replication cost and server load than sequential selection. The breakdown of misses shows that random selection has significantly more connection misses than sequential selection. This is because the

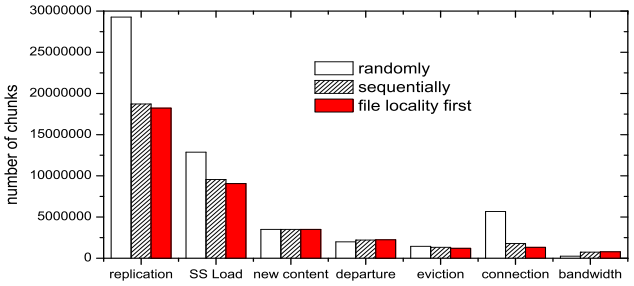

Figure 5: Lazy-oracle under different configurations. Here, the x-axis replication label shows the total number of chunks replicated between peers. SS Load means the load of source servers, calculated by the total number of source fetches. A source fetch is a miss. SS Load is divided into the rest five categories by cause, new content, peer departure (departure), peer eviction (eviction), connection issue (connection), and insufficient bandwidth (bandwidth), as described in Section 2.2.

chunks of a video are replicated to many different peers. When a later joined peer wants to play the same video, it needs to create more connections to get the scattered chunks from many other peers. Due to constraints on the number of connections for each peer, the scattered chunks cause connection misses. Motivated by this observation, we propose a target peer selection policy using file locality, called *file locality first*. With this policy, a peer first chooses one of the target peers that cache the same file but save different chunks. If the peer cannot find any peer with the same file from the target peer list, it picks the first one, following the rule of sequential selection. Figure 5 shows that this policy achieves the fewest connection misses, the lowest server load and number of replications.

In the lazy replication framework, the *lazy factor* (α) is a tunable parameter that limits the upload capacity used by replication. With a smaller α, more chunk replication will be delayed until when the peer is predicted to leave. Figure 6 presents the performance of lazy-oracle with different lazy factors. Note that a higher α reduces a few departure misses, yet at much higher cost. Although it decreases some server load, it increases other kinds of misses, such as eviction misses, connection misses and bandwidth misses. In the end, the improvement in server load is not signficant. When the lazy factor increases from 0 to 1, the efficiency significantly decreases. This result shows that lazy replication with a lower lazy factor is more efficient.

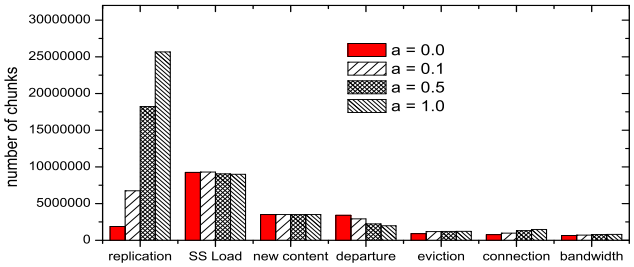

Figure 6: Lazy-oracle under different lazy factors (α). The x-axis labels are explained in the caption of Figure 5.

4.2.2 Performance Comparisons

The above results show that lazy-oracle can achieve the best performance, when parameterized by a lazy factor of

zero and uses a *file locality first* policy. Using this configuration, the following experiments compare lazy-oracle to eager replication and examine the gap between lazy-oracle and lazy-simple.

In the experiment of eager replication, K is set to 1, the same value as in lazy replication. Figure 7 compares lazy-oracle to eager replication. Eager replication reduces more departure misses than lazy-oracle, but has a higher source server load. The total number of replicated chunks in lazy-oracle is 45% of that in eager algorithm. The cost of lazy-oracle is much lower than that of eager replication. In terms of the decrease of server load, lazy-oracle is better than eager replication. Taken together, the efficiency of lazy-oracle is 0.78, more than three times that of eager replication (0.21). These results suggest that lazy-simple can replicate more efficiently than eager replication.

Figure 7 shows that lazy-simple fairly approximates to lazy-oracle in terms of server load. Both decrease server load by 15% from GridCast without replication. Compared to lazy-oracle, lazy-simple replicates more chunks and has a higher cost. That is because the chunk request predictor tends to overestimate the popularity (described in Section 3.2.2). In lazy-simple, more chunks can be selected into the candidate chunk list to be replicated. Although they help lazy-simple to reduce more departure misses than lazy-oracle, the overall server load does not decrease. Compared to eager replication, lazy-simple decreases more server load at a lower cost. Finally, the efficiency of lazy-simple (0.33) is still higher than that of eager replication (0.21). This shows again that lazy-simple outperforms eager replication.

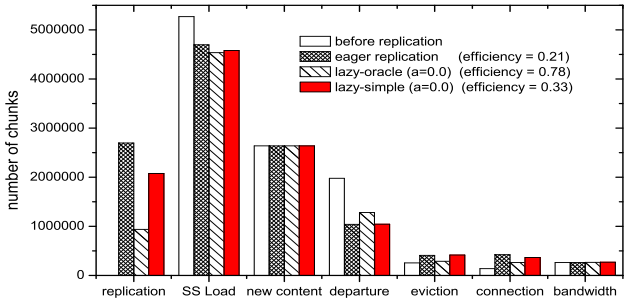

Figure 7: Comparison of Lazy-simple, Lazy-oracle, and Eager.
The x-axis labels are explained in the caption of Figure 5.

5. RELATED WORK

Many studies on P2P VoD have been done in the past few years, but most of them focus on topology optimization [4] [7] and caching algorithms [3] [8]. Vratonjic et al. [10] discuss proactive caching in a P2P VoD system, named BulletMedia, which combines a traditional overlay mesh approach with a structured overlay to efficiently support block dissemination and forward seeks. Allen et al. [1] examine LRU and LFU caching algorithms for VoD services in a relatively stable environment, such as cable networks. They investigate the performance of LRU caching in detail. Recently, Huang et al. [8] explore the potential of peer-assisted sharing in VoD through trace-driven simulation. All of these previous work only focus on peer sharing in a single video, not considering the substantial locality effects in a multiple video environment.

Bhagwan et al. [2] present some discussions on replication in a storage system, called TotalRecall. They discuss how to predict the availability of host components based on their past behavior and take the appropriate redundancy mechanisms and repair policies. Similar to this work, we employ some predictors to assist replication. Differently, our study focuses on chunk replication in a highly dynamic environment, particularly in a P2P VoD system. Those unique features of VoD, such as short session duration and random seeks, make our issue more challenging. For example, our results show that an eager replication seems less efficient than a lazy replication in this system.

6. CONCLUSIONS

Based on the off-line log analysis of GridCast, a deployed P2P VoD system, we identify that departure misses become a big issue in a P2P VoD system with caching optimization. Motivated by this observation, we examine how to use replication to decrease departure misses and further reduce server load. We design and evaluate a lazy replication framework based on two predictors; a peer departure predictor and a chunk request predictor.

Through trace-driven simulation, we demonstrate that lazy replication with two simple and feasible predictors can further decrease about 15% of server load, compared to the real GridCast system. In addition, we compare lazy replication to a naïve eager replication algorithm. The simulation results show that lazy replication with two simple predictors is more efficient than eager replication in a dynamic P2P system VoD like GridCast.

7. REFERENCES

[1] M. Allen, B. Zhao, and R. Wolski. Deploying Video-on-Demand Services on Cable Networks. In *Proc. of ICDCS*, 2007.

[2] R. Bhagwan, K. Tati, Y. Cheng, S. Savage, and G. M. Voelker. Total Recall: System Support for Automated Availability Management. In *Proc. of NSDI*, 2004.

[3] M. Cha, H. Kwak, P. Rodriguez, Y.-Y. Ahn, and S. Moon. I Tube, You Tube, Everybody Tubes: Analyzing the World's Largest User Generated Content Video System. In *Proc. of IMC*, 2007.

[4] B. Cheng, H. Jin, and X. Liao. Supporting VCR Functions in P2P VoD Services Using Ring-Assisted Overlays. In *Proc. of ICC*, 2007.

[5] B. Cheng, X. Liu, Z. Zhang, and H. Jin. A Measurement Study of a Peer-to-Peer Video-on-Demand System. In *the 6th International Workshop on Peer-to-Peer Systems*, 2007.

[6] B. Cheng, L. Stein, H. Jin, and Z. Zhang. Towards Cinematic Internet Video-on-Demand. In *Proc. of EuroSys*, 2008.

[7] Y. Guo, K. Suh, J. Kurose, and D. Towsley. P2Cast: Peer-to-peer Patching Scheme for VoD Service. In *Proc. of WWW*, 2003.

[8] C. Huang, J. Li, and K. Ross. Can Internet Video-on-Demand be Profitable. In *Proc. of SIGCOMM*, 2007.

[9] J. Tian and Y. Dai. Understanding the Dynamic of Peer-to-Peer Systems. In *IPTPS*, 2007.

[10] N. Vratonjic, P. Gupta, N. Knezevic, D. Kostic, and A. Rowstron. Enabling DVD-like Features in P2P Video-on-demand Systems. In *SIGCOMM Peer-to-Peer Streaming and IP-TV Workshop (P2P-TV)*, 2007.

[11] H. L. Yu, D. D. Zheng, B. Y. Zhao, and W. M. Zheng. Understanding User Behavior in Large-Scale Video-on-Demand Systems. In *Proc. of EuroSys*, 2006.