THROUGHPUT DRIVEN, HIGHLY AVAILABLE STREAMING STORED PLAYBACK VIDEO SERVICE OVER A PEER-TO-PEER NETWORK

K.Kalapriya, S.K.Nandy

{kalapriya,nandy}@cadl.iisc.ernet.in
CADL, Indian Institute of Science

ABSTRACT

Streaming video is a key functionality in Internet based services such as distant education, VoD etc.,. The high bandwidth requirements associated with such applications coupled with limited I/O bandwidth of the server limits the throughput levels achieved by these systems. Such systems can largely benefit if the abundant resources such as storage and bandwidth available at the end systems (otherwise called as peers) are put to use. Availability of Streaming service is largely affected by the transient nature of the peers. Therefore, we identify two important issues in these systems,viz (i) Scalability,(in terms of support large number of clients) (ii) High Data availability (minimizing the loss of segments due to the transience of peers). In this paper we propose a throughput driven, highly available 'streaming stored video' service architecture over a peer-to-peer network. We use distributed caching scheme where the initial segments of data are temporarily cahced and served to subsequent requests, thereby reducing the number of requests to the server. We exploit the inherent redundancy of FEC based channel coding to provide high availability. Our aim is to show that despite this high transience of peers, the proposed distributed caching scheme helps reduce the number of server streams requested and provide high data availability, while serving large community of clients. We describe our three-tier architecture to realize the proposed model and measure the potential of our scheme in terms of increase in overall system capacity.

1. INTRODUCTION

Internet based services like video broadcasts, distance learning etc., require streaming of video to multiple users simultaneously. Streaming video reduces the startup time of the video by simultaneously playing the video while downloading segments of video as opposed to the traditional download and play systems. Currently streaming video architecture is a two-tier architecture consisting of streaming entity and a set of requesting clients. In such systems, the overall system capacity (number of simultaneous streams served) is limited by the I/O bandwidth and network resources at the server. One of the main issues is the scalability of the streaming service, which is very less compared to the number of user requests that span the Internet. We define the scalability as support for large number of users. Though deploying proxies and caches at several locations can increase the system capacity, it multiplies the overall system cost and introduces many administrative complexities such as cache consistency and load balancing problems.

Peer-to-Peer architecture seems to be a probable fit for streaming video applications that can take advantage of the abundant resources (I/O and network bandwidth and storage resources) available at the end-systems. Designing a 'streaming stored video' [1]architecture over a peer-to-peer network should strive to provide high data availability despite this high transience nature.

In our work we consider 'peer-to-peer' architecture for 'streaming stored video' and identify two important issues (i) scalability and (ii) high data availability. We use distributed and temporal caching of initial segments of a playback video after which the stream can be treated as near live stream. While a given peer continues to receive segments of the stream, simultaneously caching these segments locally renders this peer to in turn act as server for other requests. We exploit the inherent redundancy of the FEC based channel coding to retrieve the stream without loss of video segments caused due to the transience of peers. We propose a three-tier architecture consisting of the server, service providing entity and client for guaranteeing highly available streaming service. The Service providing entity admits clients based on the availability criteria submitted by the clients at the time of service request. We run simulations to measure the efficiency our scheme in terms of increase in overall system capacity while providing data availability.

Rest of the paper is organized as follows. Section 2 presents the related work, Section 3 presents the system overview and the scheme proposed. Section 4 presents the results. We conclude our work in Section 5.

2. RELATED WORK

In this section we review the literature in streaming video that are closely related to our work.

Broadcast techniques for streaming video has been discussed in [1],[2],[3],[4]. These techniques have been categorized into reactive transmission approach, proactive transmission approach and hybrid approach. In these approaches

[1]In 'streaming stored video' the server starts a new stream for every request. We differentiate 'streaming live video' from 'streaming stored video', by the fact that every request in "streaming stored video" should mimic a start of new stream, whereas in "streaming live video" the clients receive the streams from the time of request.

Proceedings of the 19th International Conference on Advanced Information Networking and Applications (AINA'05)
1550-445X/05 $20.00 © 2005 IEEE

a single server stream serves several request for the same video arriving spaced closely in time. In these techniques though the server can serve large communities, the throughput is still limited by the I/O, network bandwidth at the server.Periodic broadcast schemes have been proposed as an alternate to broadcast in [5], [6]. A video segment is fragmented into a number of segments and each segment is periodically broadcasted on a dedicated channel. In these schemes, if the client misses the beginning of the video stream, the clients have to wait until the next broadcast session thereby increasing the latency of the playback.

Chaining [5], introduced the peer-to-peer(P2P) streaming paradigm, by caching portions of video data. Clients can forward the video to other downstream clients, reducing the load on the video server. In our scheme we merge chaining and patching for playing stored playback video where the end-clients are used for streaming the patch [initial segments]. Our work benefits from utilization of the I/O and network bandwidth available at the peers thereby conserving resources at the server.Streaming video over P2P has been addressed in the context of live streaming video [6][7][8]. Our work extends beyond live streaming video to 'stored playback video' over P2P network where each request is served from the beginning of the stream.

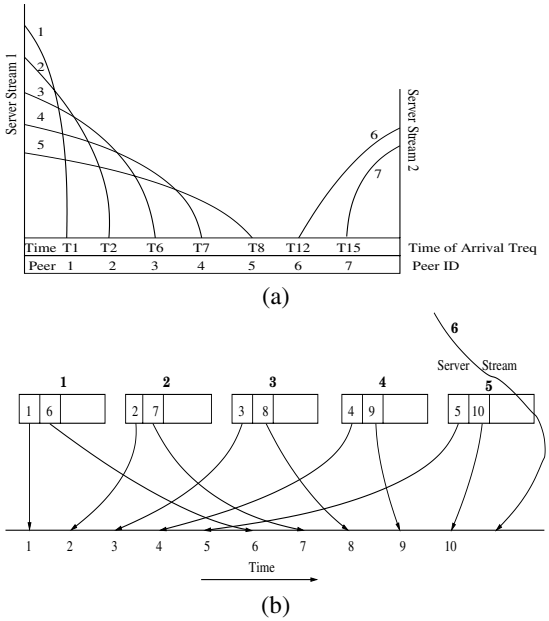


Fig. 1. (a) Video stream from the server serving multiple peers (b) Initial segments of video being served by peers to a single request at time T10

3. ARCHITECTURE OF STREAMING STORED VIDEO OVER PEER-TO-PEER NETWORKS.

In case of streaming stored video the server will serve every individual request as a new stream. The number of streams served in these systems is directly proportional to the bandwidth available at the server. Network resources like bandwidth and storage available in abundance at the end systems can be used to overcome the problem of bandwidth saturation at the server. These end-systems or peers while receiving the stream can also help in distributing the stream to other peers. We describe the transmission of video segments in the following section.

3.1. Distributed caching among Peer

Transmission of video segments is modeled as a discrete time model (T) at the granularity of video packets. Every peer receiving the stream contributes buffer of size C_{psize} in accordance with the availability of its memory.

On arrival of a peer the total cache (C_{tot}) increases by the size of the cache contributed by the peer and is expressed as

$$C_{tot} = C_{curr} + C_{psize}$$, where C_{psize} is the cache contribution of the peer C_{curr} is the total cache currently availble. C_{tot} is the total amount of cache buffer/blocks available.

Without loss of generality C_{tot} can be mapped to the discrete time scale T as T_{cmax}. We use T_{cmax} and C_{tot} interchangeable since T_{cmax} and C_{tot} correspond to buffer size in time and space respectively.

Any peer joining within the time interval T_{cmax} (mapped to C_{tot}) can be served by the existing stream.

Figure 1(a) gives a logical view of peers being served by a stream. The total available cache blocks (C_{tot}) is 8. Peers 1,2,3 and 4 at intervals T1, T2, T6, T7, T8 are served by a single server stream (server stream 1) forming a cluster. A 'gap' is formed between the intervals 8 and 12. Peers 6,7 arriving at time T12 and T15 is served by a new server stream (server stream 2) and forms a new cluster. Figure 1(b) shows detailed view of fetching of the initial segments from the peers and being served by the stream. Peers 1,2,3,4 and 5, has cached the segments up to T10. If Peer 6 arrives at T11, it could fetch the initial segments from the peers and receive the stream continuously. This requires extra buffer on the peer to receive the continuous stream while downloading initial segments of the stream. A cluster is said to be in stabilized state when a gap is formed *i.e.,* the same stream can serve no more peers. In the above example P5 acts as transit node for the stream for P6.

The above scheme leads to two important issues.
(i) Exit of a peer that contributes the buffer for caching can lead to the formation of 'Gap'(loss of video segments) with a probablity

$$P_{gap} = \frac{C_{psize}}{C_{tot}} \quad (1)$$

where P_{gap} is the probability of formation of the gap. This implies that probability of formation of 'gap' is proportional to the contribution of peer to the total cache size.

(ii) On exit of a transit node[2], the receiving peers need to find an equivalent transit peer to continue receiving the stream. It takes time for the receiving peer to find an equiv-

[2]Transit node is a node that acts as router node and helps forwarding the stream to other nodes, it other words it acts as a transit node for the stream.

alent transit peer. During this transience there will be loss of video segments.

3.2. Redundancy to manage transience of Peer

During an end-host multicast session, a peer can unsubscribe in the middle of the stream, partitioning the network. The time taken to repair the partition will result in loss of segments (packets) transmitted during the transience period. Forward Error Correction (FEC) is a channel-coding scheme used to correct loss of data packets. We exploit the FEC scheme to overcome the loss of packets during this transience period.

A video stream is typically chopped off into segments, each of which is packetized into k packets; then for each segment, a block code is applied to the k packets to generate a j-packet blocks(j = k+n where n is the extra segments obtained), where $j > k$. This group of packets is transmitted to the receiver, which receives k packets. To perfectly recover a segment, a peer much receive at least k packets out of j packets that is transmitted thereby increasing the transmission rate by a factor of n/k that pose a limitation on n.

Let n denote the number of redundant segments required, n_{pexit} the number of peers that exit, k the number of segments required to decode a video segments, n_{peer} the number of peers that caches the video segments.

3.2.1. Redundancy Requirement on exit of a transit node

On exit of a node that acts as a transit node, n is bounded by the time taken to repair the partition created by the transience of peers.

$$n = C(T_{exit} - T_{rejoin}) \tag{2}$$

where $(T_{exit} - T_{rejoin})$ gives the time taken to repair the partition and $C(T_{exit} - T_{rejoin})$ is simply the mapping of discrete time intervals to video segments.

3.2.2. Redundancy Requirement on exit of a peer that contributed Cache(C_{psize})

In case of exit of peer that contributed for cache segments, 'n' is bounded by the

$$n \leq n_{pexit}.(k/n_{peer}) \tag{3}$$

to avoid forming a gap. In equation (3),

when $k = n_{peer}, n = n_{pexit}$
when $k < n_{peer}, n < n_{pexit}$
when $k > n_{peer}$, $n$is bounded by$(k/n_{peer})*$ n_{pexit}.

Combining equations (2) and (3) we obtain

$$n = max\left[C(T_{exit} - T_{rejoin}), (n_{pexit} * (k/n_{peer}))\right] \tag{4}$$

Thus adding redundancy equivalent to n given by the above expression decreases the formation of gaps,thereby reducing the number of new stream request to the server. Consequently in equation (1) if

$$C_{psize} < n, P_{gap} \approx 0. \tag{5}$$

3.3. Throughput Analysis

A cluster is being served by a single stream till the gap is formed. From eqn.(1) P_{cache}/P_{tot} imposes a gap and results in the request to a new stream from the server . Addition of 'n' decreases the formation of gap by $C_{psize}/C_{tot} * [1 - (k/k + n)]$,
where '$k/k+n'$ gives the fraction of information in terms of required video segments and $[1 - (k/k + n)]$ gives the fraction of information loss due to redundancy. Thus eqn.(1) can be rewritten as,

$$P_{gap} = \frac{C_{psize}}{C_{tot}}[\frac{k}{k+n})] \tag{6}$$

Subject to,
$0 < C_{psize}/C_{tot} \leq 1, k/(k+n) \leq 1, C_{psize} > 0, C_{tot} > 0$

From eqn.(6) it can be see that increase in 'n' reduces the formation of the gap by the probability defined by eqn.(6). In other terms the peer losses it's importance (measured in terms of cache contribution) equivalent to the amount of cache contributed by the client times amount of redundant information added to the stream. From the forgoing discussions it can be said that 'n' plays an important role in determining the throughput by reducing the formation of the gap.We run simulations to determine the throughout for various values of 'n' and different cache size contributed by the peers.

3.4. Availability Analysis

We define availability of data as complete retrieval (without loss of segments) of the stream on exit of peers. Data availability is largely affected due to exit of peers that has cached the initial segments. Redundancy introduced helps in retrieval of video without loss of segments. Thus, 'n' plays an important role in data availability on exit of a peer that is serving the initial segments to a receiving peer.

Eqn.(3) gives the amount of redundancy required to make the data available on exit of peers. It can be seen that 'n' is directly proportional to the number of peers that exit given a constant value of 'k' and cluster size. In other terms the amount of redundancy 'n', decides availability of video segments on exit of peers commenserable to 'n'. We derive the expression for determining the availability criteria below.

Let P_{avail} be the availability of the client, T_{peer} be the duration of the peer and T_{stream_time} is the duration of the stream, then availability of the node for the given session is determined by

$$P_{avail} = T_{peer}/T_{stream\_time.} \tag{7}$$

Eqn.(7) gives the importance of peer measured in terms of its availability. Exit of a peer affects the stream (loss of video segments) by the factor '$(1 - P_{avail})$'. For any given peer, if $P_{avail} \approx 0$, this implies that the peer affects the availbility of video segments significantly. To minimize the loss of video segments on exit of peers whose $P_{avail} \approx 0$,

the peer is admitted with increased value of 'n'(more redundancy).Given different values of 'n', our aim is to determine the average size of the cluster and throughput by admitting peers proportional to its availability criteria given by eqn.(7). We run simulations for different values of 'n' and determine its effect on throughput. We describe our three tier architecture for admitting the peers and guaranteeing data availability for these peers in the next section.

4. THREE-TIER ARCHITECTURE FOR CLIENT ADMITTANCE

In this section we propose a three-tier [3] architecture for realizing our proposed scheme. The Three-tier model comprises the streaming server, the receiving clients and a middleware service providing entity (SPE). The SPE has two main functions (i)admittance control that decides on the cluster to which the requesting client joins to receive the ongoing stream (ii) Decides the equivalent peer on exit of a transit node. Equivalent peer is a peer node that can stream the data to the receiving peer from the time 't' of exit of transit node. The admission control protocol and Equivalent peer finding algorithm is described in the following subsections.

4.1. Admittance Protocol

Server Agreement: From eqn.(3), it can be seen that 'n' required for retreiving the stream by the peers is directly proportional to the number of peers that exit. The SPE decides the value of 'n' [4] with the server. This value of 'n' determines the sustenance [5] of the stream for number of peers that exit proportional to 'n'. This implies that even on exit of peers proportional to 'n', the stream can be retreived without loss of segments. Thus SPE admits clients based on this value of 'n' and the availability criteria submitted by the client. Admittance of client is a three step protocol as shown in fig.(2(a))

Step 1: Request from the client.

The client sends a request to the SPE to receive the streaming video. The request encapsulates its credential like exepected time of participation, bandwidth available and cache size contribution by the peer.

Step 2: Decision algorithm.

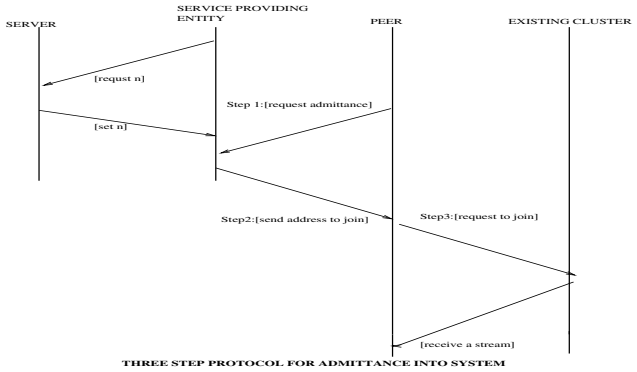
The SPE decides to admit the client to an existing cluster or request for a new stream depending on the availability criteria given by eqn.(7). Fig.(2(b)). gives the algorithm that SPE uses to decide the admittance of clients into the system.

Step 3: Return of Selected cluster (Peer address) The SPE returns the peer address got from step 2 to which the requesting node should join for receiving the stream. If the request resulted in a new stream, the requesting peer forms the root node of the stream. Subsequent peers receives the

[3] We use this architecture to simulate the P2P streaming environment for measuring throughput.

[4] We measure the throughput for various values of n.

[5] Recovery of stream without loss of data.



(a)

```
Return_addr()
{
    N= get_N_afterset();
    get_client_credential();
    pavail =calculate_pavail();
    boolean= map_pavail_availability();
    while(client_arrival()){
            if(Boolean is set && N>0)
               admit_client();

             If(Boolean is not set && N>0)
               { admit_client();
                 N--;
               }
             if(N>0)
             {
              N= get_N_afterset();
              Open_new_cluster(send request to server);
              admit_client();

              N--;
             }
        }
}
```

Algorithm for admittance of client into the system

(b)

Fig. 2. (a) Three step protocol for admittance of client. (b) Algorithm for determining the appropriate cluster

stream from on-going stream resulting in the formation of application layer multicast chainging tree. This multicast tree helps in constructing the dependency graph to identify an equivalant peer and is described below.

4.1.1. Application Layer Multicast Chaining Tree

The multicast chaining tree is modelled as directed tree T= (V,E) , where V= 0,1,2..n is the set of nodes representing peers and E is the set of directed edges within the tree. Consider a node $i \in v$. Let $p(i)$ denote its parents and $s(i)$ and set of children attached to node i i.e.,$(p(i), i) \in E$. Associated with every node is cache of capacity $c(i)$,$0 < c(i) < \infty$. Any node i that initiates a stream by requesting the server becomes the head of the cluster and root of the multicast tree. Any subsequent node j joins the multicast chaining tree with root i. The SPE maintains the $\{ident(j), tj, c(j)\}$, where ident(j) is the identity of the node j and t(j) the time it has joined the multicast tree and c(j) its capacity.

4.2. Equvivalent Peer Discovery

On exit of a root transit node, the receiving peer has to find an equivalent peer. Given 'T',the time exit of peer our aim is to find out the peer that would allow the receiving peer to join and receive the stream with minimal overhead. The

SPE uses the multicast chaining tree(s) and creates a dependency graph to determine its set of predecessors and successors. The dependency graph is a directed graph consisting of nodes and directed edges. The cost of edge is the time lag between the nodes measured in terms of cache units available. Eqn(8 and 9) gives the rules for determining the set of predecessors and successors.

$$Set\ of\ Predecessors = T_{req} < T_{node} + T_{csize} \quad (8)$$

$$Set\ of\ Successors = T_{req} > T_{node} \quad (9)$$

where T_{req} is the time of request from the receiving peer, T_{node} is the current time of the already receiving node, T_{csize} is the cache size, equivalent to the time difference between T_{req} and T_{node}. Fig.(3) shows a typical dependecny graph [6].

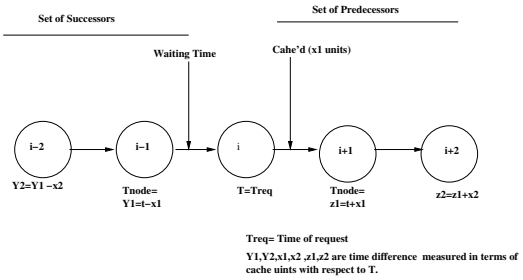


Fig. 3. Dependency graph formed from the chaining tree

Given this set of predecssors or successors an equivalent node is decided such that

$$Equivalent\ node = node[min(Waiting\ time, T_{csize})] \quad (10)$$

Eqn.(10) reduces the overhead in terms of waiting time when a successors is chosen and buffer requirement on the node when predecessor is chosen. We present our simulation results in the next section.

5. RESULTS

We use discrete time based simulation for studying effects of various cache sizes and duration of peers. We measure the following performance parameters i) The number of streams served from the server.ii) Throughput: The number of peers served (limited capacity server).iii) Effect of 'n' on throughput.iv) The number of Clusters formed (and the average size of the cluster) when data availability is guaranteed. For simulation we use the following fixed parameters:

i)A media file of 30 min duaration recorded at CBR rate of 100 kb/s. ii) Server mimics the midstream video server that supports upto 250 simultaneous streams iii)Simulation time fixed to 4 hrs

To simulate a flash crowd we consider high request rates about 9 requests/min, 15 requests/min and 30 requests/min.

[6]Creating dependency graph is advantageous during exit of multiple peers,where the edges can be directed from the already existing peers nodes thereby reducing the time for finding another equivalent peers

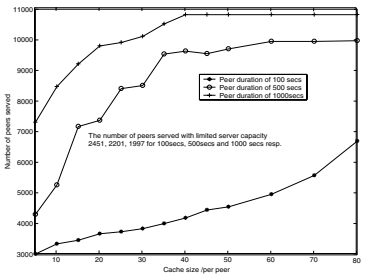


Fig. 5. Throughput in terms of number of peers served for different duration of peers

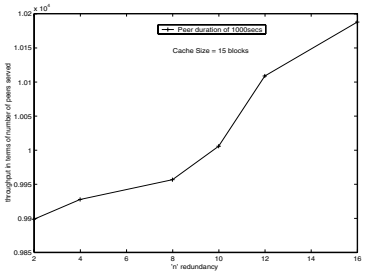


Fig. 6. Effect of n on throughtput

To consider the transient nature of peers we simulate peer groups of different durations (100 secs, 500 secs and 1000secs). Data sets were collected for varying cache size contributed by the peers (Cache of size 5 ,10 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 70, 80 blocks).

Fig.(4(a) and (b)) plots the number of streams requested from the server [7]for different duration of peers and mean rates respectively. It can be observed that even when contribution of client is small (order of 5 to 25 units) the number of streams requested is reduced by 65% approximately.

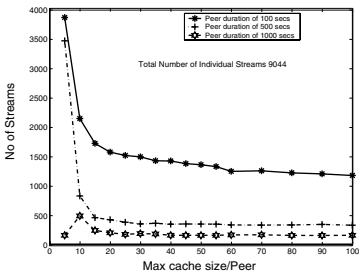
Fig. (5) plots the throughput in terms of number of peers served. It can be seen that the throughput increases sharply for smaller values of cache size and becomes constant for increased value of cache sizes. This is due to the fact the increased cache size results in increased penalty on exit of peers limiting the throughput at higher values (of about 25 units). This implies that small caches sizes by themselves greatly compliment the streaming video system.

Fig.(6) plots the throughput for increasing values of 'n' for fixed value of 'k'(cache size contribution for each peer = 15, duration of peers= 1000secs). It can be observed that when n=2, throughput is increased by 6.77% . This implies that adding 'n', decreases the probability of formation of gap (by the factor of 1/n). But increasing 'n' increases the rate of transmission consuming bandwidth.

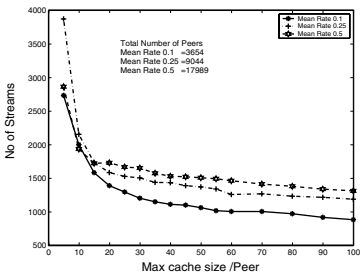
Fig (7(a)) and (7(b)) measures the throughput[8] in terms of number of clusters served on a limited server capacity. When 'n' is increased the number of clusters reduces with increase in cluster size (no of peers served by a single stream.)

[7]It is assumed that the server is of unlimited capacity for measuring the number of streams requested from the server.

[8]The data sets was collected with limited arrival of clients of 1428,3654,17989

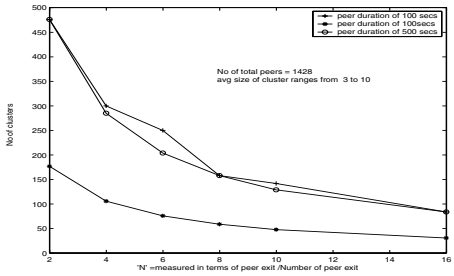


(a)

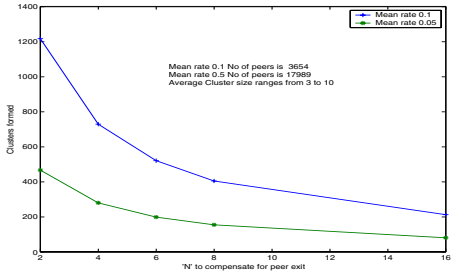


(b)

Fig. 4. (a)streams requested by the peers for different request rates (b) streams from the server for different duration of peers



(a)



(b)

Fig. 7. Throughput when availability is satisfied for (a) different arrival rates (b) for different duration of peers

The average cluster size ranges from 3 to 10. From the foregoing discussions it can be seen that using the p2p architecture for streaming stored playback video reduces the load on server to around 65% and increase in throughput on a limited capacity server (10.07% approximately) with minimum overhead on the peer nodes(around 15% for cache size of 15 blocks).

6. CONCLUSIONS AND FUTURE WORK

In our work we proposed distributed caching scheme over peer to peer architecture where the initial segments are video are cached to mimic the start of 'Playback video'.This scheme finds wide applications in internet based services like distance learning where the client requests (flash crowds) are spaced closely in time. To reduce the effect of formation of gap we exploit the inherent redundancy of the FEC based channel coding. This scheme effectively uses the resources (storage) at the end-system and helps increase the throughput of the server (approximately 10.07% increase in the number of peers served) while reducing the load (number of simultaneous streams to around 65%)on the server. We measure the effect of 'n' on throughtput and observed that there is increase of throughput (approximately 6.77% when n=2). This implies that adding redundancy greatly complements data availability. We further propose a three tier architecture (that takes advantage of the redundancy to provide data availability) that consists of a service providing entity to guarantee data availability to the clients. We are in the process implementing our prototype in terms of services offered over the Internet. Further work can be continued with formation of clusters of similiar requirements in terms of rate and quality video.

7. REFERENCES

[1] T. D. Little and D. Venkatesh, "Prospects for Interactive Video-on-Demand". IEEE Multimedia, 1(3):14–25, May 1994.

[2] H.Ma and K.G. Shin, "Mutlicast video-on-demand services," ACM Communications Review pp.31-42. Jan 2002.

[3] E.L.Abram-Profeta and K.G.Shin, "Scheduling video programs in near video on demand systems,", in Proc. ACM Multimedia '97, pp.359-369.

[4] A.Dan, D.Sitaram, and P.Shanhabuddin, "Dynamic batching policies for an on-demand video server," Multimedia Systems vol.4, pp.112-121, June 1996.

[5] " A permutation-based pyramid broadcasting scheme for video-on-demand systems," in Proc.International Conference on Multimedia Computing and Systems, 1996, pp.118-126.

[6] H. Deshpande, M. Bawa, and H.Gracia-Molina, "Streaming Live Media over a Peer-to-Peer Network". Tech.Rep. CS-2001-26, CS Dept., Stanford University, 2001

[7] Duc A. Tran, Kien Hua, Tai Do, "Peer-to-Peer Streaming Using A Novel Hierarchical Clustering", ICDCS 2003.

[8] D. Xu, M. Hefeeda, S. Hambrush, B. Bhargava, "On Peer-to-Peer Media Streaming", In Proc. of International Conference on Distributed Computing Systems (ICDCS'02), Vienna, Austria, July 2002.