ISSN:2320-0790

An International Journal of Advanced Computer Technology

Design of Distributed Prefetching Protocol in Push-to-Peer Video-on-Demand System

Thiruselvan. R

Assistant Professor
Computer Science & Engineering Department
Sri Venkateswara college of Engineering & technology
Affiliated to Anna University, Chennai

Abstract: Peer-to-peer networks have to streaming the video on the Internet. In P2P streaming system, the upstream bandwidth of peers are larger than video playback rate. This system does not overcome the upstream bandwidth limitation by server based stream delivery. But, the push-to-peer system does not rely on content servers except in the push phase. So, this system can overcome the bandwidth limitation. In this paper is content placement and associated pull policies that allow the optimal use of uplink bandwidth in push-to-peer video-on-demand system. The initial content placement increases the content availability and improves the use of peer uplink bandwidth. The mostly required videos are proactively pushed to the set-top-boxes in the digital subscriber line networks during time of low network utilization that is in the early morning. There are two approaches used for content placement and pull policies, which are Full striping scheme and Code-based placement scheme. The client can easily download and play out video from set-top-boxes. So, it would reduce server load, network load and downloading time. In Full striping scheme, videos are strip into video blocks and push the distinct video block into the set-top-box. This system has to provide high quality of video streaming and to reduce the client's waiting time. This system can reduce the server load, network load and congestion. In Code-based placement scheme, videos are encoded into coded symbol by using rate less code algorithm. This approach could eliminate the box failure in full striping scheme. When the client's required data is not available in the set-top-box, then the distributed prefetching protocol used to directly connect client to the video server and streaming the video from video server to client.

Key words: Full striping scheme, Code-based placement scheme, Upstream bandwidth, Distributed Prefetching Protocol, Set-top-box, video streaming, Push phase, pull phase etc.

I. INTRODUCTION

A .VIDEO-ON-DEMAND SYSTEM

Video-on-Demand (VOD) systems either stream content through a set-top box, allowing viewing in real time, or download it to a device such as a computer, digital video recorder, personal video recorder or portable media player for viewing at any time. The majority of cable and telephone company based television providers offer both VOD streaming, such as pay-per-view, whereby a user buys or selects a movie or television program and it begins to play on the television set almost instantaneously, or downloading to a digital video recorder rented from the provider, for viewing in the future.

Download and streaming video on demand systems provide the user with a large subset of VCR functionalityincluding pause, fast forward, fast rewind, slow forward, slow rewind, jump to previous/future frame etc. These functions are called trick modes. For disk-based streaming systems which store and stream programs from hard disk drive, trick modes require additional processing and storage on the part of the server, because separate files for fast forward and rewind must be stored. Memory-based VOD streaming systems have the advantage of being able to perform trick modes directly from RAM, which requires no additional storage or CPU cycles on the part of the processor. It is possible to put video servers on LANs, in which case they can provide very rapid response to users. Streaming video servers can also serve a wider community via a WAN, in which case the

responsiveness may be reduced. Download VOD services are practical to homes equipped with cable modems or DSL connections.

B. PEER-TO-PEER NETWORK

A Peer-to-Peer (P2P) computer network uses diverse connectivity between participants in a network and the cumulative bandwidth of network participants rather than conventional centralized resources where a relatively low number of servers provide the core value to a service or application. P2P networks are typically used for connecting nodes via largely ad hoc connections. Such networks are useful for many purposes. Sharing content files containing audio, video, data or anything in digital format is very common, and real time data, such as telephony traffic, is also passed using P2P technology.

A pure P2P network does not have the notion of clients or servers but only equal peer nodes that simultaneously function as both "clients" and "servers" to the other nodes on the network. This model of network arrangement differs from the client-server model where communication is usually to and from a central server.

C. PUSH-TO-PEER SYSTEM

In this paper, we have to design of a *Push-to-Peer* Video-on-Demand (VOD) system. In such a system, video is first pushed (e.g., from a content creator) to a population of peers. This first step is performed under provider or content owner control, and can be performed during times of low network utilization (e.g., early morning). Note that as a result of this push phase, a peer may store content that it itself has no interest in, unlike traditional pull-only peer-to-peer systems. Following the push phase, peers seeking specific content then pull content of interest from other peers, as in a traditional peer-to-peer system. The Push-to-Peer approach is well-suited to cooperative distribution of stored video among set-top-boxes in a DSL network [14], where the set-top boxes themselves operate under provider control. We believe, however, that the Push-to-Peer approach is more generally applicable to cases in which peers are long-lived and willing to have content proactively pushed to them before video distribution among the cooperating peers begins.

We begin by describing an idealized policy for placing video data at the peers during the push phase - full striping and its consequent pull policy for downloading video. We also consider the practical case in which the number of peers from which a peer can download is bounded, and propose a code-based scheme to handle this constraint. We demonstrate that these two placement policies are optimal among policies that make use of the same amount of storage per movie, in that they maximize the demand that the system can sustain. The remainder of this paper is structured as follows. In Section II, we describe Literature Survey. In Section III, we describe two policies for placing video data at the peers during the push phase. In Section IV described about distributed prefetching protocol in Push-to-peer System.

II. LITERATURE SURVEY

The peer-to-peer concept has been applied to more general video-on-demand services. P2P networks for streaming video on the Internet have generated a lot of interest recently. P2P-based streaming systems completely rely on peer connections, which make the system vulnerable to peer or connection failures. Then combine P2P techniques with the current server-client streaming model to build a hybrid system that is both scalable and robust. First, propose a streaming system, BitTorrent Assisted Streaming System (BASS) proposed by Danna. et al., [4] for VoD services, where we add the use of an external streaming server to a slightly modified BitTorrent. Clients can simultaneously stream from the media server as well as each other via BitTorrent P2P connections. By maintaining these connections, we can reduce the aggregate bandwidth used by the media server and decrease client waiting times.

BitTorrent Streaming (BiToS) proposed by Vlavianos. et al., [6] a protocol with the ability to support streaming based on BiTorent. We identify the piece selection mechanism as the only thing that needs to be changed from the original BiTorrent protocol. BiToS becomes aware of the streaming order of the piece, thus preferring pieces that will be played soon. BitTorrent has been proved to be a very effective mechanism for P2P content distribution. The success of BiTorent lies on its ability to distribute content quickly by utilizing the capacity of all the peers in the P2P BT network.

A prefix caching technique proposed by Sen. et al., [5] whereby a proxy stores the initial frames of popular clips. Upon receiving a request for the stream, the proxy initiates transmission to the client and simultaneously requests the remaining frames from the server. In addition to hiding the delay, throughput, and loss effects of a weaker service model between the server and the proxy, this is simple prefix caching technique aids the proxy in performing workahead smoothing into the client playback buffer. By transmitting large frames in advance of each burst, workahead smoothing substantially reduces the peak and variability of the network resource requirements along the path from the proxy to the client. The proxy prefix caching technique for peer-to-peer video streaming require upstream bandwidth of a peer to be larger than video playback rate.

However, most of the efforts have focused on efficient tree and mesh construction, assuming the upstream bandwidths of peers are larger than video playback rate. Under this assumption p2p systems can scale to support arbitrarily large numbers of clients. In contrast, we can cope with uplink bandwidths smaller than video playback rate, a condition that holds in most access networks, particularly DSL. More recently, Tewari. et al., proposed BitTorrent [7] based live streaming service under the same assumption of limited upstream bandwidth. In both proposals, the upstream bandwidth limitation is overcome by the assistance of server based stream delivery in their proposed systems. However, the Push-to-Peer system does not rely on content servers except in the push phase.

Our proposed scheme collectively balances all sub requests for a job. Another related area of work is the data placement and pull scheme for video streaming services. Several methods have been proposed in the literature. Particularly, random duplicated assignment strategy of data blocks and mirroring are proposed for VOD servers by Korst and Bolosky et al.,[10] respectively to address the problem of disk failure. However, we use a code-based placement that addresses the problem of box failures Rateless coding algorithm has been proposed by Maymounkov. et al., [3],[11],[12]. While these works discuss how to use the codes to download files using multicast/broadcast transmissions or using peer-to-peer networks, none of these works address the usage of coding for video streaming or video-on-demand. Other work proposed the use of network coding to accelerate file download in peer-to-peer networks or to ameliorate VOD for P2P.

Recently Jin and Bestavros proposed a scalable "cache-and-relay" approach [8] that could be used for scenarios similar to the one motivated above. Using this approach, a recipient of the feed would "cache" the most recently played out portion of the feed (after playing it out). Such cached content could then be used by other nodes in the system who request the feed within some bounded delay. This process of caching and relaying the content was shown to scale well in terms of server as well as network loads. In [9], a detailed analysis of this approach was presented There are two problematic aspects of the cache-and-relay approach. First, when a node leaves the system, any other nodes receiving the feed from that node are disconnected. This means that such nodes will experience a disruption in service. Second, to resume, such discon nected nodes must be treated as new arrivals, which in turn presents added load to the server (and network). This latter issue is especially significant because recent results by Jin and Bestavros [13] have shown that asynchronous multicast techniques do not scale as advertised when content is not accessed from beginning to end (e.g., due to nodes prematurely leaving the multicast and/or when non-sequential access is allowed to support VCR functionality). Specifically, Jin and Bestavros showed that techniques that ensured asymptotic logarithmic server scalability under a sequential access model would in effect behave polynomially under non-sequential access models

Prefetch-and-relay protocol for scalable asynchronous multicast in P2P systems proposed by Sharma. et al., [2] that allows a peer to serve as a source for other peers, while prefetching a portion of the stream ahead of its play out time in Peer-to-Peer system. In contrast to existing cache-and-relay schemes, our scheme is more scalable in highly dynamic Peer-to-Peer systems. This is because a departure of a peer does not necessarily force its children peers (for whom it is serving as source) to go to the original server. Rather a child peer can continue its play out uninterrupted from its prefetched data until it is covers a new source peer. The download rate is sufficiently greater than the play out rate; our distributed prefetching scheme significantly reduces the load on the server as it effectively increases the capacity of the Peer-to-Peer system. At the same time, clients can achieve a better play out performance. More importantly, a client can proactively switch from one source-peer to another in order to reduce the transmission delay of its download or to optimize the overall network link cost. We have include that distributed prefetching protocol [2] concept in Push-to-Peer Video-on-Demand system [1]. The Push-to-Peer System design and analysis model is proposed by K. Suh. Et al.,[1]. This system described by two approaches, which are full striping scheme and code-based placement scheme.

III. DATA PLACEMENT AND PULL POLICIES

In this section we first propose the full striping data placement and code-based data placement schemes. In contrast to full striping, the latter allows a box to download a video from a small number of boxes. This is useful when the number of simultaneous connections that a box can support is constrained. VCR operations such as jump forward, jump backward, and pause can be supported by both schemes.

A. Full Striping Scheme

A full striping scheme stripes each window of a movie overall M boxes. Specifically, every window is divided into M blocks, each of size W/M, and each block is pushed to only one box. Consequently, each box stores a *distinct* block of a window. A full window is reconstructed at a particular box by concurrently downloading $M - 1$ distinct blocks for the window from the other $M - 1$ boxes. Hence a single movie download request generates $M - 1$ sub-requests, each targeted at a particular box.

A box serves admitted sub-requests according to the Processor Sharing (PS) policy, forwarding its blocks of the requested video to requesting boxes. PS is an adequate model of fair sharing between concurrent TCP connections, when there is no round-trip time bias and the bottleneck is indeed the upstream bandwidth. We further impose a limit on the number of sub-requests that a box can serve simultaneously. Specifically, to be able to retrieve the video at a rate of R_{enc}, one should receive blocks from each of the $M - 1$ target boxes at rate at least R_{enc}/M, where R_{enc} is the video encoding/playback rate. Hence we should limit the number of concurrent sub-requests being served by each box to at most $K_{max} := B_{up}M/R_{enc}$, where B_{up} is the upstream bandwidth of each box. This approaches for handling new video download requests that are blocked because one of the $M - 1$ required boxes is already serving K_{max} distinct sub-requests.

B. Code-Based Placement Scheme

We describe a modification of full striping, namely code-based placement, under which the maximum number of simultaneous connections that a box can serve is bounded by y, for some $y < M - 1$. This scheme applies rateless coding [3], [11].This rateless coding algorithm is describe in section C. A rateless code such as the LT code [11] can

generate an infinite number of so-called coded symbols by combining the k source symbols of the original content. The code-based scheme, we have divides each window into k source symbols and generates Ck = (M /(y +1))k coded symbols. We call C is the expansion ratio, where C > 1. For each window, the Ck symbols are evenly distributed to all M boxes such that each box keeps Ck/M = (1 + ϵ)k/(y + 1) distinct symbols. A viewer can reconstruct a window of a movie by concurrently downloading any Cky/M distinct symbols from an arbitrary set of y boxes out of (M − 1) boxes. The code-based scheme is similar to full striping in the sense that distinct (coded) symbols are striped to all M boxes. However, unlike full striping, only y boxes are needed to download the video. We now define the pull strategy used for the code-based scheme. The maximum number, K_{max}, of sub-requests that can be concurrently processed on each box to ensure delay free playback now reads K_{max}= $(y + 1)B_{up}/R_{enc}$. Under the blocking model, a new request is dropped, unless there are y boxes currently handling less than K_{max} sub-requests. In that case, the new request creates y sub-requests that directly enter service at the y boxes currently handling the smallest number of jobs. Under the waiting model, each box has a queue from which it selects sub-requests to serve. Each new movie download request generates M −1 sub-requests that are sent to all other boxes. Upon receipt at a receiving box, each sub-request either enters service directly, if there are less than K_{max}sub-requests currently served by that box. Otherwise it is placed in a FIFO queue specific to the box. Once a total of y sub-requests have entered service, all other M − 1 − y sub-requests are deleted. Thus each request eventually generates only y sub-requests.

C. Rateless Code Algorithm

This section explains how to implement on-line codes. A more detailed description and analysis of the algorithm is available in [3],[11],[12]. On-line codes are characterized by two parameters ϵ and q (in addition to the block size). ϵ determines the degree of sub optimality a message of n blocks can, with high probability, be decoded $(1+3\epsilon)n$ from output blocks. The first step of the encoding process is to produce a composite message by generating $0.55q\epsilon n$ auxiliary blocks and appending them to the original message. Each auxiliary block is computed as the XOR of a number of message blocks, chosen as follows We first seed a pseudo-random generator in a deter ministic way. Then, using the pseudo-random generator, for each block of the original message, we chose q auxiliary blocks, uniformly. Each auxiliary block is computed as the XOR of all message blocks we have assigned to it. We append these auxiliary blocks to the original message blocks, and the resulting $n'=(0.55q\epsilon+1)n$ blocks form the composite message.

We call the message blocks that were XORed to produce a check or auxiliary block its adjacent message blocks. Decoding consists of one basic step: Find a check or auxiliary block with exactly one unknown adjacent message block and recover the unknown block (by XORing the check block and all other adjacent message blocks). Repeat this step until the entire original message has been recovered.

IV. PREFETCHING PROTOCOL DESCRIBTION

In this section described about distributed prefetching protocol in Push-to-peer System. Whenever the client required video is not available in set-top-boxes, then its establish connection to video server. Then, this protocol prefetch and store the content ahead of their play out time from video server. One client is streaming video from video server at that time any other client request that same video to box, then requesting client is connected to streaming client. If any content missing during time of streaming, then simultaneously

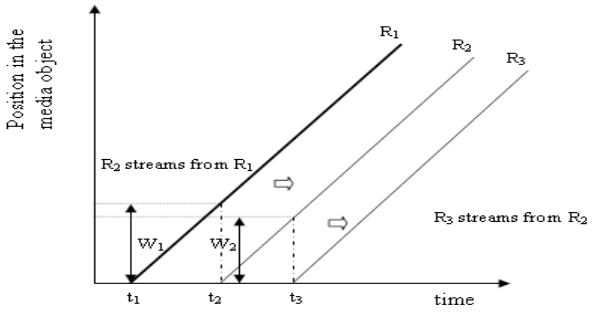

Figure 1 Asynchronous Streaming

establish another connection to the server.Assume that each client is able to buffer the streamed content for a certain amount of time after playback by overwriting its buffer in a circular manner. As shown in Fig. 1, *R*1 has enough buffer to store content for time length *W*1; i.e. the data cached in the buffer is replaced by fresh data after an interval of *W*1 time units. When the request *R*2 arrives at time $t = t2$, the content that *R*2 wants to download is available in *R*1's buffer and, hence, *R*2 starts streaming from *R*1 instead of going to the server. Similarly, *R*3 streams from *R*2 instead of the server. Thus, in Figure 1, leveraging the caches at end-hosts helps to serve three clients using just one stream from the server.

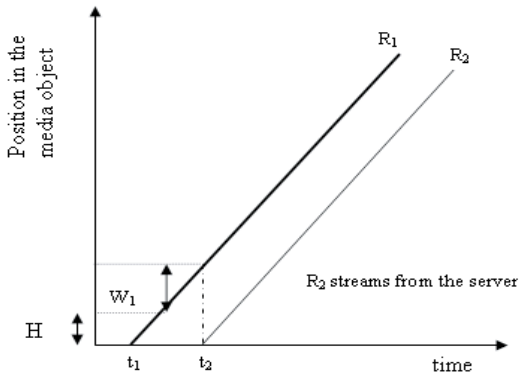

Figure 2 Asynchronous Streaming

In Figure 2, by the time request *R*2 arrives, part of the content that it wants to download is missing from *R*1's buffer. This missing content is shown as H in Figure 2. If the download rate is the same as the playout rate, then *R*2 has no option but to download from the server. However, if the network (total) download rate is greater than the playback

rate, then *R*2 can open two simultaneous streams - one from *R*1 and the other from the server. It can start downloading from *R*1 at the playback rate (assuming that *R*1's buffer is being overwritten at the playback rate 1) and obtain the content H from the server. After it has finished downloading H from the server, it can terminate its stream from the server and continue downloading from *R*1. This stream patching technique used to reduce server bandwidth. Assuming a total download rate of α bytes/second and a playback rate of 1 byte/second, the download rate of the stream from the server should be $\alpha-1$ bytes/second. Hence, for this technique to work $\alpha-1 \geq 1 \Rightarrow \alpha \geq 2$. Hence, we need the total download rate to be at least twice the playback rate for stream patching to work for a new arrival. In the event that a client departs from the peer-to-peer network, all the clients downloading from the buffer of the departing client will have to switch their streaming session either to some other client or the server. The stream patching technique can be used by a client in this situation as well to avoid downloading from the server. The stream patching technique may work in this situation even when the total download rate is less than twice the playout rate, i.e. $\alpha < 2$.

When the download rate is greater than the playoutrate, a client can pre-fetch content to its buffer before it is time to playout that content. Pre-fetching content can help achieve a better playout quality in overlay multicast. In a realistic setting, there would be a certain delay involved in searching for a peer to download from; for example, consider the situation depicted in Fig. 3.

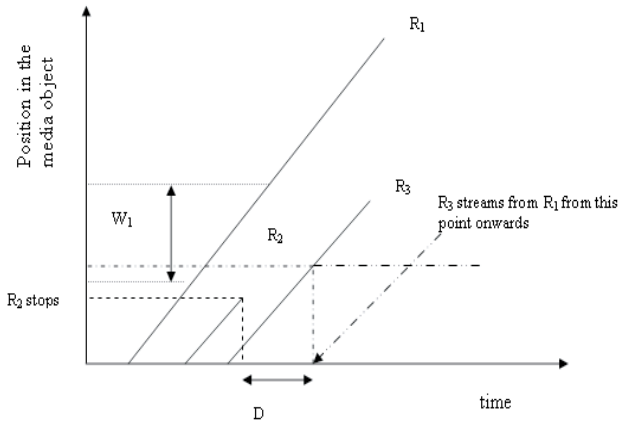

Figure 3 Delay in finding new download source

*R*3 starts streaming from *R*2 on arrival. After *R*2 departs, as shown in Fig. 3, it takes *R*3 **D** seconds (time units) to discover the new source of download *R*1. If the pre-fetched "future" content in *R*3's buffer, at the time of *R*2's departure, requires more than **D** seconds (time units) to playout (i.e. the size of the future content is greater than **D** bytes, assuming a playout rate of 1 byte/second) then the playout at *R*3 does not suffer any disruption on *R*2's departure. If the size of the "future" content is smaller than **D** bytes, then *R*3 will have to open a stream from the server, after it has finished playing out its pre-fetched content, till it discovers *R*1. if the time required to playout the pre-fetched content is larger than the delay involved in finding a new source to download from, the playout at *R*3 would not be disrupted upon *R*2's departure from the peer-to-peer network. Pre-fetching content is also advantageous when the download rate is variable. A client can absorb a temporary degradation in download rate without affecting the playout quality if it has sufficient pre-fetched content in its buffer.

A. *Control Parameters*

In this paper, we analyze the importance and effect of the following three parameters in achieving scalable (in terms of server bandwidth), asynchronous delivery of streams in a peer-to-peer environment.

1) $\alpha = \dfrac{\text{Download rate}}{\text{Playout rate}}$

Without loss of generality, we take the *Playout rate* to be equal to 1 byte/second and, hence the *Download rate* becomes α bytes/second. We assume $\alpha > 1$.

2) *Tb* : The time it takes to fill the buffer available at a client at the download rate.

The actual buffer size at a client is, hence, $\alpha \times Tb$ bytes. The available buffer size at a client limits the time for which a client can download the stream at a rate higher than the playout rate.

3) $\beta = \dfrac{\text{Future content}}{\text{Past content}}$

β represents the ratio of the content yet to be played out, "future content", to the content already played out, past content", in the buffer.

B. *Constraints in the case of an arrival*

For a new arrival *R*0 to be able to download from the buffer of *R*1, the inter-arrival time between *R*0 and *R*1 should be less than *Tb*. If *R*0 arrives more than *Tb* time units after *R*1, then part of the content that *R*0 wants to download would have been over-written in *R*1's buffer. If $\alpha < 2$, then in such a situation *R*0 has no option but to stream from the server. Hence, if $\alpha < 2$, a new arrival at time $t = t0$ can stream from only those clients that arrived during the interval $TD = [t0-Tb, t0)$. If $\alpha \geq 2$ and *R*1 is over-writing the content in its buffer at the playout rate, then *R*0 can take advantage of the higher download rate (compared to the playout rate) and the stream patching technique to possibly avoid a complete streaming from the server and instead download from *R*1 and only patch the missing content from the server. It is easy to verify that the size of the missing content that *R*0 needs to download from the server, in order to be able to stream from *R*1's buffer, cannot be greater than the size of the available buffer at *R*0, which is $\alpha \times Tb$ in our model. A newly arrived client *R*0 can download from the buffer of *R*1 if the following conditions are satisfied:

- The inter-arrival time between *R*0 and *R*1 is less than *Tb*, or
- If the inter-arrival time between *R*0 and *R*1 is greater than *Tb*, then α should be greater than or equal to 2, *R*1 must be over-writing the content in its buffer at the playout rate and the size of the

content missing from *R*1's buffer should be less than or equal to $\alpha \times Tb$.

The first condition ensures that the content needed by *R*0 is present in *R*1's buffer. The second condition defines the scenario in which the stream patching technique can be used by *R*0.

V. RESULTS AND DISCUSSIONS

We have implemented Push-to-peer VOD System using java with JMF. We have using socket programing for implementation of video server and client module and then set-top-boxes implemented by oracle9i database. This system is implemented by two approaches which are full striping scheme and code-based placement scheme. The important videos are proactively stored on set-top-boxes. Whenever video is require to client which is recontructed in target box from all other boxes. If, the required video is not available in set-top-box, then set-top-box manager establishing connection between video server and client and streaming video. That video prefetch and store ahead of play out time is done by using RTP and RTCP protocol in JMF.

In full striping scheme, the number of requests in progress varies from box to box, because a requesting box does not place a request on it self. Also, the overall system service speed varies between (M-1)B_{up} and MB_{up} depending on System state: when a single video download takesplace, it proceeds at speed (M-1)B_{up}, while an overall service rate of MB_{up} is achieved when sub-requests are served on all boxes. The number of sub-request is same on all box, and the total service capacity is also constant. Specifically, we consisder a total service capacity of B_{total}=MB_{up} and assume this is stored evenly among active downloads. Denote by Lj is the size of movie j in bytes and by Aj,m is amount of memory in bytes dedicated to movie j on box m. We shall assume that a single copy of each movie is stored in system, which can be translated into the constraint $\Sigma_{m=1\ M}=L_j$.

In code-based placement scheme, we assume that a total storage capacity of C*Lj is denoted to movie j , where C=M*(1+ ϵ)/y+1 is expantion ratio. The solution based on encoding assumes that for movie j , a total quantity of $A_{j,m} \equiv C*L_j/M$ data is stored on each individual box m. This data consist of symbols, such that for any collection of y+1=M/C boxes, each movie can be reconstructed from the joint collections of symbols from all these y+1 boxes.

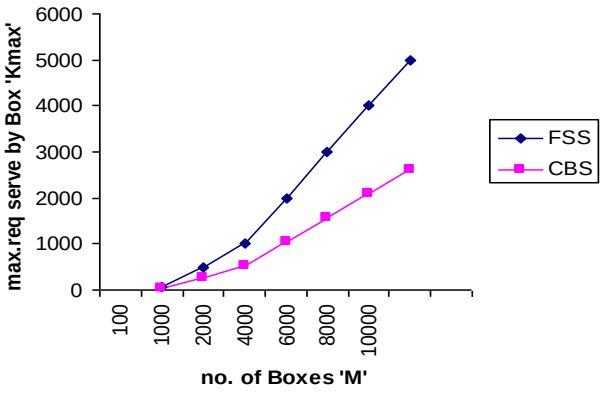

Figure.4 Maximun number of sub-request served by each box in FSS and CBS Vs total no of boxes 'M' with video encoding/play back rate Renc = 2 Mbps, upstream bandwidth Bup = 1 Mbps, coding overhead ϵ = 0.05 and C=2.

Figure 4 shows that maximum number of sub-request serve by each box in FSS and CBS approaches are varies with respect to number of set-top-boxes. These are described that FSS allows viewers to take advantages of bandwidth from all M boxes regardless of number of served viewers, while CBS constrains number of boxes. Also, the service rate are equally share in FSS approach because all boxes used for each movie request . So, the service speed at FSS approch is less than CBS service rate. Because CBS approach required constrains minimum boxes compared to FSS approach. So, the waiting time of client in CBS is more less than the system in FSS approach.

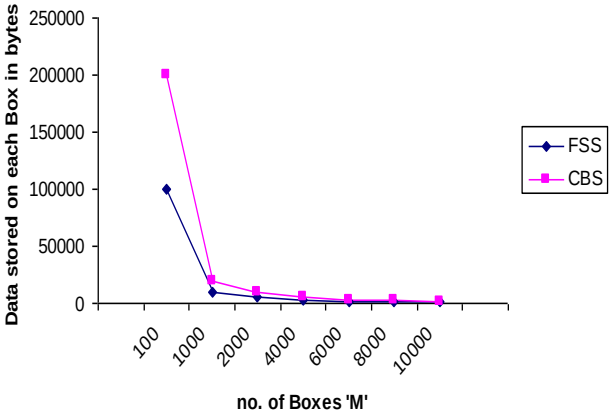

Figure.5 Content stored on each box in FSS and CBS Vs total no of boxes 'M' with size of video Lj = 10 Mbytes and C = 2.

Figure 5 shows that amount of memory in bytes dedicated to movie j on box 'm' is varies with respect to number of boxes increases in FSS and CBS approach. The CBS pushes 20 copies of videos to 10 boxes collectively. On the other hand, the FSS pushes only one copy of the video. The FSS consistently outperforms the CBS, even thougth the last scheme benefits from langer amount of data stored on each boxes.

VII. CONCLUSION AND FUTURE DIRECTION

Push-to-Peer Video-on-Demand System has implemented by the two approaches with distributed prefetching protocol in which the two approaches are full striping scheme and code-based scheme. The video server proactively pushed the important video into set-top-boxes in client premises that in DSLAM in a DSL network. The boxes are served the required video to clients perfectly. Then, the prefetching protocol streamed the video to client, whenever the client required video is not available in set-top-boxes. This system could reduce client waiting time, server load, network congestion and network load.

This project will be extended by client to client streaming using prefetching protocol, while the same video is downloading from video server or another client. If any content missing during streaming from another client or departure of that client, then it's simultaneously establish a

connection to server and then download the missing content from server.

VI. REFERENCES

[1] K. Suh, C. Diot, J. Kurose, L. Massouli´e, C. Neumann, D. Towsley, and M.Varvello., "Push-to-peer video-on-demand system: design and evaluation", IEEE Journal in Communications, volume 25, issue 9, pp. 1706-1716, December, 2007.

[2] A. Sharma, A. Bestavros, and I.Matta., "dPAM: a distributed prefetchingprotocol for scalable asynchronous multicast in P2P systems", IEEE Conference on Computer Communication, Boston University, USA, pp. 1-13, 2006.

[3] P. Maymounkov and D. Mazieres. "Rateless codes and big downloads", International Workshop on Peer-to-Peer Systems, Newyark University, pp. 11-16, February 2003.

[4] C. Dana, D. Li, D. Harrison, and C. Chuah., "BASS: BitTorrent assisted streaming system for video-on-demand", IEEE seventh workshop on Multimedia Signal Processing, 2005.

[5] S.Sen, J. Rexford, and D. Towsley., "Proxy prefix caching for multimedia streams", IEEE Conference on Computer Communication, pp. 1310-1319, 1999.

[6] A. Vlavianos, M. Iliofotou, and M. Faloutsos., "BiToS: Enhancing BitTorrent for supporting streaming applications", IEEE Global Internet Symposium, Barcelona, 2006.

[7] S. Tewari and L. Kleinrock., "Analytical model for BitTorrent video streaming", IEEE National Institute of Multimedia Education Workshop, pp. 976-980, 2007.

[8] S. Jin and A. Bestavros, "OSMOSIS: Scalable Delivery of Real-Time Streaming Media in Ad-Hoc Overlay Networks", in Proceedings of IEEE ICDCS'03 Workshop on Data Distribution in Real-Time Systems, 2003.

[9] Y. Cui, B. Li, and K. Nahrstedt, "oStream: Asynchronous Streaming Multicast in Application-Layer Overlay Networks", IEEE Journal on Selected Areas in Communications, 22(1), January 2004.

[10] J. Korst, "Random duplicated assignment: An alternative striping in video servers", in Proceeding of ACM Multimedia, 1997.

[11] M. Luby., "LT Codes", IEEE Symposium on foundation of computer science (FOCS). IEEE Computer Society, 2002.

[12] Petar Maymounkov, "Online Codes", Technical Report TR2002-833, New York University, October 2002.

[13] S. Jin and A. Bestavros, "Scalability of Multicast Delivery for Non-sequential Streaming Access", in Proceedings of SIGMETRICS'2002: The ACM International Conference on Measurement and Modelling of Computer Systems, 2002.