Dynamic Distributed Collaborative Merging Policy to Optimize the Multicasting Delivery Scheme*

X.Y. Yang[1], Porfidio Hernández[1], F. Cores[2], A. Ripoll[1], R. Suppi[1], and Emilio Luque[1]

[1] Computer Science Department, ETSE, Universitat Autònoma de Barcelona 08193-Bellaterra, Barcelona, Spain
[2] Computer Science & Industrial Engineering Department, EPS Universitat de Lleida, 25001, Lleida, Spain

Abstract. The advance of Internet 2 and the proliferation of switches and routers with level three functionalities made the multicast one of the most feasible video streaming delivering techniques for the near future. Assuming this to be true, this study addressed the over-load situation that a streaming server could suffer due to client requests. As a solution, we proposed new multicast delivery scheme that allows every active client to collaborate with the server regardless of the video that they are watching, alleviating server loads, and therefore server resource requirements. The solution combined the multicast delivery scheme and client-side buffer collaboration in order to decentralize the delivery process. The new video delivering scheme was designed as two separate policies: the first policy used client collaboration to deliver first part of videos and the second policy could merge two or more multicast channels using distributed collaboration between a group of clients. Experimental results show that this scheme is better than previous schemes in terms of resource requirements and scalability.

1 Introduction

The high increase in the commercial use of the Internet (distance learning, Video on Demand (VoD) and digital video libraries) has generated a substantial growth in the demand for video streaming systems. In video streaming environments, users request the videos they desire and a server delivers the requested video information; allocating, using the most simple delivery technique, a dedicated server unicast channel for each video request. Even though the unicast delivery scheme is easy to implement, it is excessively expensive and there is a lack of scalability.

In order to reduce the cost of video-delivery and attain high server scalability, three complementary research approaches have been investigated: (1) server

* This work was supported by the MCyT-Spain under contract TIC 2001-2592 and partially supported by the Generalitat de Catalunya- Grup de Recerca Consolidat 2001SGR-00218.

J.C. Cunha and P.D. Medeiros (Eds.): Euro-Par 2005, LNCS 3648, pp. 879–889, 2005.

transmission schemes using multicast, this strategy allows users to share server and network bandwidth to reduce the individual service cost; (2) video streaming technique with application layer multicast enables multicast transmission schemes beyond a local area network, assuming only IP unicast at the network layer; and (3) proxy caching [6], enabling high scalability for clients dispersed across a wide-area. The main focus of this study is the design of multicast delivery in order to reduce the individual service cost, specially, we proposed a delivery scheme that is able to offer true VoD services[10].

Sophisticated video delivery techniques based on multicast have appeared such as Batching [4], Patching [1][2], Adaptive Piggybacking [3], Merging[5], Chaining [7] and Cooperative Video Cache(CVC)[8].

With a Batching technique, video requests for the same video that are submitted in the same short interval time are served by a single multicast channel. Clients suffer a certain period of waiting time and the average length of waiting time depends on the policy of selecting the clients to serve with the first available channel. Due to this waiting time, a Batching approach only provides near-VoD service. A Batching approach is also called static multicast since late coming requests are not allowed to join any already on-going multicast channel. With Patching, however, clients are dynamically assigned to join multicast channels. Since late coming clients miss part of the video information, a separate unicast channel, called a patch stream, is needed to deliver the first part of the video. The Patching approach assumes that clients can simultaneously download two streams and has a local buffer, capable of saving t minutes of video. While a client is watching a video from the patch stream, the video information arriving from multicast channels is buffered. Even though the Patching policy provides true-VoD service, the server resource requirement increases depending on the request arrival frequency due to unicast channels. Furthermore, a request is only able to join a multicast channel if the difference between the request arrival time and the multicast channel start-time is lower than t.

Like Patching, Adaptive Piggybacking and Merging are also dynamic multicast approaches. In the Piggybacking policy, the server slows down and speeds up the delivery rate of two consecutive multicast channels in order to merge two multicast channels into one. The number of channels that Piggybacking can merge is limited by the fact that less than 5% adjustment of the delivery rate is allowed, in order to preserve the display quality that clients receive. The Merging policy, however, does not change the display quality. Two multicast channels are merged using the client buffer. In the Merging policy, while clients are playing the video, they try to buffer video information from a previous multicast channel. This policy can only merge channels that are started in a period of time no longer than the length of video information that each client is able to save in their buffer.

The main ideas behind Chaining and CVC are fairly similar. Both policies are based on the creation of a delivery chain in which video information is forwarded from one client to another. With these policies, a new client receives the video from an existing chain and does not consume any server bandwidth. However, delivery chains could only be formed if the interarrival times of client requests

are short, limited by the size of each individual client's buffer. Furthermore, only clients that are watching the same video can take part in the formation of the chain.

In this paper, we propose a new delivery technique called Dynamic Distributed Collaborative Merging (DDCM). The DDCM technique is based on the peer-to-peer paradigm and allows every active client to collaborate with the server regardless of the video that they are watching. The client collaborations are performed by two complementary delivery policies. Under the first policy, while successive incoming requests are allowed to join an existing multicast channel, the missed video information (patch stream) is delivered by another client who is playing the same video. Unlike the Patching policy, the patch stream does not consume the server's resources. The aim of the second policy is to dynamically merge multicast channels using distributed buffers. More than one client of different videos could be used in the merging process of two channels. The merge policy is able to merge multicast channels regardless of the time between their start-times. The merge policy enables clients of unpopular videos to help the server to merge the channels of more popular videos, and vice versa.

The rest of the paper is organized as follows: in section 2, we show the key ideas behind DDCM. Performance evaluation is shown in section 3. In section 4, we indicate the main conclusions of our results and future work is explained in the final section.

2 Dynamic Distributed Collaborative Merging Scheme

In the delivery scheme design, we assume that clients are able to hold two symmetric channels. We assume that video information is encoded with Constant Bitrate (CBR) and that each client channel is able to receive/send one video stream. We refer to network unicast channel that delivers the first part of a video as patch stream and the multicast channel that delivers the information for the complete video as complete stream.

The DDCM delivery scheme is designed as two separate policies: 1) Patch Stream Manager (PSM) whose main role is to deliver patch streams using client collaborations. 2) Complete Stream Manager (CSM). The main function of this second policy is to try to merge two or more complete streams into one.

2.1 Patch Stream Manager Design

When the first request from client C_i arrives time t_i, the server opens a new complete stream($M1$) for the client. When a second request from client C_{i+1} arrives in time t_{i+1}, the server decides whether or not the client can be served by using a previous complete stream ($M1$). In order to serve by using a previous complete stream, client(C_{i+1}) must have enough buffer to save more than $(t_{i+1} - t_i)$ seconds of video information from the complete stream. If not, the server will open a new complete channel. In the other case, a patch stream is needed to send video information from 0 to $(t_{i+1} - t_i)$. The remaining video information $((t_{i+1} - t_i)$ to the end) will be sent by the previous complete stream.

For each new patch stream, the PSM policy searches for an active client that has the first part of the video in their local buffer. In such a case, a collaborative client will open a patch stream and send the first part of the video. Client C_{i+1} will also join the previous multicast complete stream. Should there be no such client, the server starts a new patch stream using server bandwidth.

Fig 1 shows the delivery process following PSM for 6 clients. Each client has 3 minutes of buffer and client requests arrive at minutes 1, 2, 3, 5, 6, and 7. Under the clients name, we indicate the length of buffer that each client is dedicated for the collaboration. For example, C3's collaboration buffer is 1 minute, while C2 collaborates with a buffer of 2 minutes. These values depend on the length of the patch stream that the client needs for the delivery process. In the case of C1, no patch stream is needed, so the full buffer (3 minutes) is used for collaboration. In order to know the buffer size that each client dedicates to collaboration, each client sends a control message to the server when the client has filled the buffer with the first part of video.

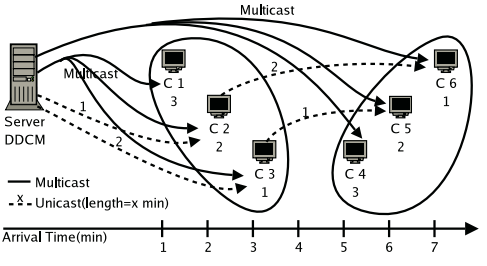


Fig. 1. Patch Stream Manager

Fig 1 shows that C2 and C3 are served using one multicast channel and 2 patch streams using server bandwidth. In the case of C5 and C6, the patch streams are delivered by C3 and C2 respectively. As we can see in Fig 1, the PSM policy is capable to deliver patch streams without consuming the server's bandwidth after minute 5.

The more clients are accepted by the server, the more client collaborations will be produced with PSM policy. This characteristic makes the PSM especially suitable as a delivery scheme for highly demanded video where a lot of patch streams are needed. However, after several minutes, the server has more than one client that is able to serve the same patch stream. This redundancy implies poor client resource utilization since many of clients will not be involved in the collaboration mechanism of PSM.

2.2 Complete Stream Manager Design

The CSM's aim is to merge the existing complete streams. Once a complete stream is merged into another, the complete stream will not consume any server resources. Since complete streams are usually long, and therefore demand most of

the server's resources, the CSM efficiently replaces the server resource demand with client collaborations. The CSM scheme achieves a high degree of client resource utilization since almost every client is involved in the collaboration mechanism regardless of the video that they are watching.

Given two multicast channels (M1 and M2), the key idea of CSM is that a group of clients form a collaborative buffer to merge M2 with M1. Then, the multicasting channel ($M2$) from the server is replaced by a channel ($M2^1$) from the collaborative clients. Since more than one client could be used in the merging process, the CSM is able to merge multicast channels regardless of the time between their start-times.

Fig 2 shows the collaborative buffer created between clients C1, C2, C3 and C6 that collaborate by providing 3, 2, 1 and 1 minutes of buffer respectively. A total of 7 minutes of video information could be saved in this collaborative buffer.

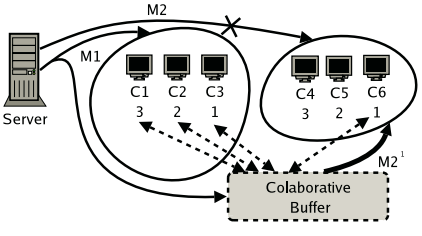


Fig. 2. Complete Stream Manager

Each client of the collaborative group successively saves video information from M1 and then delivers the information to M2 when clients of this channel need it. Fig 3 shows the delivery process of two multicasting channels($M1$ and $M2$). The $M2$ was started 4 minutes(S) later than $M1$. The merging process starts when block C and 8 are being delivered to $M1$ and $M2$ respectively. Channel $M2$ is closed in time 4, and a new channel $M2^1$ is opened.

In order to know what information each client has to save, the video information of M2 is divided into blocks and enumerated. The CSM decides the list of blocks that a client has to save. For example, $C1$ saves blocks [C,D,E], [J,K,L] and so on. After saving the blocks [C,D,E] in time 0-3, $C1$ waits 1 minute before starting to send the video information to $M2^1$. Block E is sent to $M2^1$ in time 6 and, after that, the $C1$ starts to save blocks [J,K,L].

Once the channel $M2$ is merged with $M1$, the CSM has to guarantee that while a client is delivering video blocks, there are enough other clients that are saving other video blocks being delivered by the server with $M1$. Since, each client can only use one stream in the collaboration, either to deliver or to save video information, the two processes (delivery and saving) have to be performed separately. In the case of Fig 3, while C4 is delivering block I, it should not receive any video information except the video that C4 is playing.

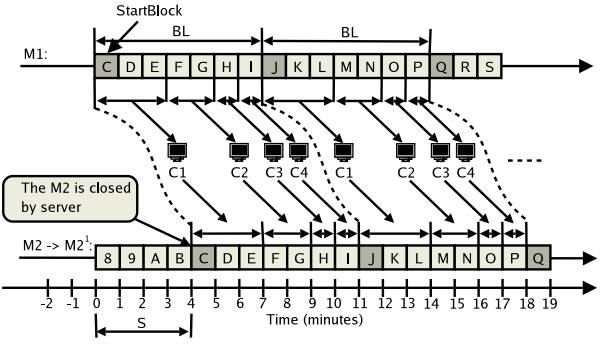


Fig. 3. CSM delivery process

2.3 Client Collaboration Group Construction Process

In the merging process, two parameters are determined by the CSM: 1)The client collaboration (B_{C_i}) that is the size of buffer of each client C_i that is to be dedicated to the formation of the collaborative buffer. 2) Accumulated buffer size that is the total size of the collaborative buffer. The value of these two parameters is determined under 2 constraints: a) A client cannot use more buffer than it has. b) A client only use one channel in the collaboration process.

Constraint a) is trivial and requires no further explanation. We established two conditions for the CSM group construction process in order to satisfy constraint b). Supposing that the CSM is interested in merging two channels that are separated into S units of time. We can formulate these 2 conditions as follows: Given a collaboration group CG of clients $\{C_1, C_2, ..., C_n\}$ [1] in which the buffer collaborations for each client are $\{B_{C_1}, B_{C_2}, ..., B_{C_n}\}$, the CSM has to satisfy:

1. Maximum collaboration: the collaboration (B_{C_i}) of a client C_i can not be greater than the value of S.

$$B_{C_i} \leq S \qquad \text{for all } C_i \in CG \tag{1}$$

2. Minimum accumulated buffer size: The total accumulated buffer size(BL) has to be bigger or equal ($S + \max\{B_{C_i}\}$).

$$\left(\sum_{C_i \in CG} B_{C_i} = BL \right) \geq S + \max\{B_{C_i}\} \text{ for all } C_i \in CG \tag{2}$$

Satisfying conditions (1) and (2), unconditioned by S, we get:

$$BL - B_{C_i} \geq S \geq B_{C_i} \text{ for all } C_i \in CG \tag{3}$$

[1] In the selection of clients, local network connection distance is considered in order to minimize the local network overhead.

The condition (3) indicates that the accumulated buffer size of all groups except for a client C_i is always bigger than C_i's collaboration(B_{C_i}) and is bigger than S. This means that the CSM guarantees that while a client C_i is sending video information there are enough other clients saving video information from the earlier multicast channel. Furthermore, while a client C_i is saving video information, there are enough other clients sending information. Since a client does not to have save information while it is sending, or vice versa, the CSM guarantees that in the collaboration process, each client will not use up more than one channel, leaving another one for playback. In this way, the CSM constructs the collaboration group in accordance with the following steps:

Step 1: The CSM calculates S of every pair of channels which could be merged and chooses the pair with the smallest S as channels to be merged.
Step 2: Satisfying conditions (2), the DDCM forms a list of clients $\{C_1, C_2, ..., C_n\}$. In this step, the maximum collaboration of each client C_i is limited by the condition (1).
Step 3: Blocks of video (Vb_j) that a client C_i has to save and deliver are determined by: $(Vb_j - StartBlock) \bmod BL \geq \sum_{m=1}^{i-1} B_{C_m}$ and $(Vb_j - StartBlock) \bmod BL < \sum_{m=1}^{i} B_{C_m}$ where $BL = \sum_{i=1}^{n} B_{C_i}$ (the total accumulated size of collaborative buffer), B_{C_i} is the collaboration of client C_i and $StartBlock$ is the block number which indicates the starting point of the merging process.

3 Performance Evaluation

We have used our prototype to evaluate the performance of the DDCM. There are three key questions that we are interested in addressing: 1) how much reduction in server bandwidth could be achieved using DDCM in accordance with the video's popularity? 2) How much server bandwidth is required using DDCM when the system is offering more than one video? 3) How could the client collaboration following the DDCM scheme help in a high-demand situation?

The DDCM is implemented in our prototype using C++ language under Linux system. We have implemented the entire necessary client feature in a Xine player plug-in[9]. In the experimentation, clients are emulated using a cluster of PCs and client requests are generated following a Poisson($P_k = \frac{\lambda^k}{k!} \cdot e^{-\lambda}$)process. The Zips-like($P_x = \frac{1}{x^z \cdot \sum_{i=1}^{S_v} \frac{1}{i^z}}$) distribution is used in order to assign the popularity of videos. We assume that the video length is 90 minutes and clients is able to save up to 5 minutes of video information.

3.1 Server Bandwidth Requirement According to Video Popularity

Fig 4 shows the server bandwidth requirement, in number of streams, using Patching, merging, PSM and DDCM(PSM+CSM). We perform this experiment under various request rates, which are normalized as the number of requests arriving during 90 minutes (video length). The resource requirement of a Merging

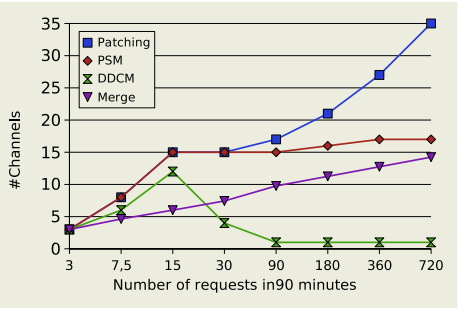


Fig. 4. Server Bandwidth Requirement for one video

policy is determined by results from [5]. We should point out that no buffer constraint is considered in Merging policy and, in a real case scenario, the Merging policy could only merge two streams separated by no more than client buffer length, so the performance will not be as good. The key observations from Fig 4 are:

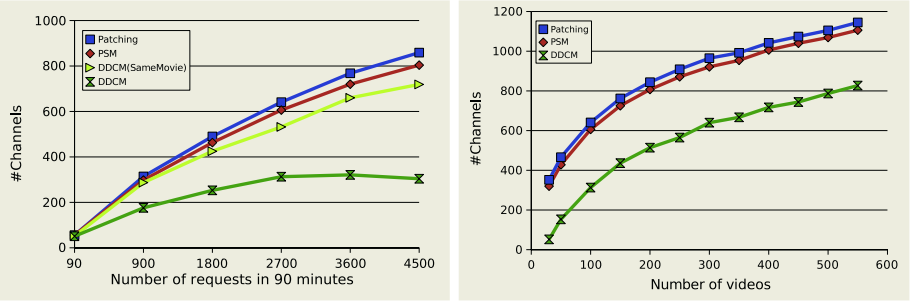
1) Using Patching policy, the bandwidth requirement increases with more requests. This makes the Patching policy unsuitable for a high demand video service.

2) Under PSM, clients can collaborate with the server to deliver patch streams. Regardless of the interarrival time, the server does not need any more than 18 streams to serve a video. This makes the PSM more suitable than a Patching policy for serving popular videos.

3) The main virtue of the DDCM(PSM+CSM) could be summarised as, 'more request less server bandwidth'. As we can see in Fig 4, the service bandwidth requirement of the DDCM increases up to 12 streams. Up to this point, there are not so many client resources that can be used to merge complete streams. As soon as a critical mass of client resources are collected, the CSM tries to merge consecutive streams and the bandwidth requirement drastically drops to 1-2 streams per video. Compared with a Patching policy, the DDCM(PSM+CSM) is able to achieve a resource reduction of 73% (4 vs. 15 streams) if there are 30 requests during 90 minutes (one request per 3 minutes). Reduction of 92.5% (2 vs. 27 streams) is achieved when there are 360 requests. Comparing with Merging policy, the DDCM(PSM+CSM) does not reduce the required resource until 30 requests. However, with 180-720 requests, the Merging policy gets closer to PSM (10-14 streams) and the DDCM(PSM+CSM) reduces the bandwidth consumption up to 85.71% (2 vs. 14 streams).

3.2 Service Bandwidth Requirement for Multiple Videos

In order to measure the bandwidth requirement of a server that is offering more than one video, we suppose that the catalog is 30 to 550 videos. We consider that the number of requests that arrive during the video length(90Minutes) is from 90 (low client activity) to 4500 (high client activity).



a)100 videos, 90-4500 requests in 90min b)30-550 videos, 2700 requests in 90min

Fig. 5. Server bandwidth requirement for multiple videos

Fig 5 a) shows the server bandwidth needed to serve 100 videos. The Patching policy shows an increase in bandwidth requirement where there is high client activity (768 and 859 channels in order to serve 3600 and 4500 requests in 90 minutes). The PSM reduces the required stream up to 6.4%(804 vs. 859), while DDCM(PSM+CSM) reduce up to 64.61% (304 vs. 859).

We also obtained the bandwidth requirement if a client is not able to collaborate with the server to merge the channels that are not delivering the same video as the one that the client is playing. As we can see in Fig 5 a) (DDCM(SameMovie)) the requirement is clearly higher than the DDCM without this restriction. These results justify our delivery policy design.

Fig 5 b) shows the server bandwidth requirement according to the size of the catalog. We have supposed that 2700 requests arrive in 90 Minutes. Regardless of the delivery policy, the bandwidth requirement increases in accordance with the number of videos. The DDCM shows a requirement reduction of between 85.23%(30 videos) to 27.77%(550 videos). The DDCM is able to reduce the number of required channels by 300-344.

3.3 Circumstantial Workload Variations

In this section we are interested in measuring the server's capacity to face circumstantial workload variations. Suppose the following situation: we are designing a VoD system for 3600 clients and, for most of the time, only 50% of the clients are active. Taking the equipment cost into consideration, the VoD server could be designed for a particular, acceptable blocking probability. Most of times, the server is able to attend to all the client requests(20 requests/minute). However, in special situations, such as the Olympic Games, all 3600 clients may decide to request videos at same time (40 requests/minute). Furthermore, since the most of population is interested in this event, the video's popularity distribution could change, increasing the skew parameter of Zipf-like distribution.

Fig 6 shows the requirement variation when twice the number of client requests reach the server. With a Patching policy, the resource requirement increases by 51.58% to 34.74% depending on the skew parameters variation. As

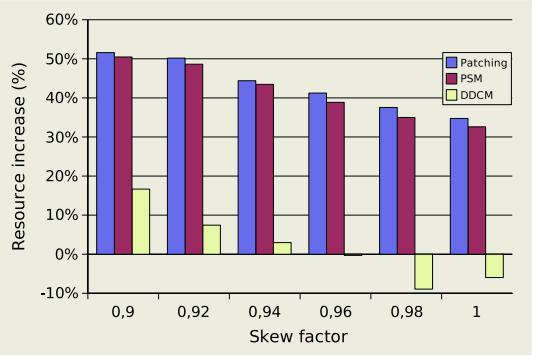


Fig. 6. Requirement increase in circumstantial workload variation

the skew fact increases, the resource requirement variation gets lower. With PSM, the resource increases 32.60% when the skew fact increases to 1 from 0.9. In this case, the PSM is 6.1% better than a Patching policy that produces an increase of 34.74%. DDCM policy produces a maximum increase of 16.67% if there is no variation in popularity distribution. In the worst case (skew fact 0.9), the DDCM is 67.68% better than a Patching policy in terms of increase in resource requirement.

4 Conclusions

We have proposed and evaluated a new video delivery technique called Dynamic Distributed Collaborative Merging that enables clients to efficiently collaborate with VoD servers. With DDCM policy, every client is able to collaborate with server, regardless of the video that they are watching. Instead of independent collaborations between the server and a client, the DDCM synchronizes a group of clients in order to merge multicast channels to achieve a better network efficiency.

Our experimental results show that DDCM has lower resource requirements than Patching policy, achieving reduction up to 92.5%. Offering multiple videos with high client activity, the DDCM is able to reduce the resource requirement up to 64.61%. These results corroborate the high scalability of DDCM when the number of requests is high. The DDCM achieves a more suitable investment in VoD server resources, since the client's punctual variation in the demand is covered by client contributions. Experimental results show that the DDCM is 67.68% better than Patching policy in terms of increase in resource requirement, suggesting that DDCM is more suitable delivery policy for VoD, in which the number of active clients changes over time.

5 Future Work

In this study, we have not considered the client network load that will suffer due to our policies and more research will be needed. However, we would like to

point out that multicast schemes are usually effective in local networks. Fault tolerance is another pending question that should be carefully analyzed.

References

1. Y. C. K. A. Hua and S. Sheu. Patching: A multicast technique for true video-on-demand services. In *ACM Multimedia Conf., Bristol, U.K.*, 1998.
2. L. Gao and D. Towsley. Threshold-based multicast for continuous media delivery, 2001.
3. L. Golubchik, J. C. S. Lui and R. R. Muntz, Adaptive piggybacking: a novel technique for data sharing in video-on-demand storage servers, in *Proc. of ACM Multimedia Systems,* 1996, pp. 140-155
4. C. C. Aggarwal, J. L. Wolf and P. S. Yu, "On Optimal Batching Policies for Video-on-Demand Storage Servers", *In Proc. of the IEEE International Conference on Multimedia Computing and Systems, Hiroshima, Japan*, June, 1996.
5. M. K. V. D. L. Eager and J. Zahorjan. Bandwidth skimming: A technique for cost-effective video-on-demand. *Multimedia Computing and Networking 2000, San Jose, CA*, 2000.
6. F. Cores, A. Ripoll, E. Luque, Double P-Tree: A Distributed Architecture for Large-Scale Video-on-Demand, *Euro-Par 2002, LNCS 2400*, pp. 816-825, Aug. 2002.
7. K. A. Hua, S. Sheu, and J. Z. Wang, *Earthworm: A Network Memory Management Technique for Large-Scale Distributed Multimedia Applications*, Proc. IEEE INFOCOM'97, Kobe, Japan, Abril 7-11, 1997, pp. 58-66
8. de Pinho, Leonardo Bidese, Ishikawa, Edison, de Amorim, Claudio Luis, *Glove: A Distributed Environment for Scalable Video-on-Demand Systems*, International Journal of High Performance Computing Applications 2003 17: 147-161
9. http://xinehq.de/, January, 2005
10. H. Ma and K. G. Shin, *Multicast Video-on-Demand Services*, ACM Communication Review, pp. 31-42, January, 2002