PROCEEDINGS OF THE INTERNATIONAL CONFERENCE ON INTERNET COMPUTING

IC'03

Volume II

Editors:
Peter Langendoerfer
Olaf Droegehorn

Associate Editors:
H. R. Arabnia
Youngsong Mun
H. A. Ramadhan

Las Vegas, Nevada, USA
June 23-26, 2003
©CSREA Press

Post-Proceedings of the International Conference on Internet Computing, Volume II, June 2003, Las Vegas, Nevada, August 2003, © CSREA Press, pp. 831-837

Server Coordinated Predictive Buffer Management Scheme to Reduce Bandwidth Requirement for Synthetic Multimedia Movies over the Internet

Arvind K. Bansal and Rahul Pathak
Department of Computer Science
Kent State University, Kent, OH 44242, USA
E-mail: arvind@cs.kent.edu and rpathak@cs.kent.edu
Phone: +1.330-672-9035, FAX: +1.330-672-7824

Abstract

Currently, for the display of synthetic and archived multimedia movies and clips, buffers are managed at the server end (push mode), at the client end (pull mode), or in the push-pull mode to avoid excessive server side disk access time, avoid hot spots for multiple disk accesses for the same media objects, and/or to smoothen the effect of variations in transmission delay. However, the current schemes suffer from the lack of static analysis and the coordination between server and the clients to reduce jitters. In this paper, we describe a server coordinated predictive buffer management scheme for mobile devices that integrates static analysis of frame based synthetic movies and server directed client side management to reduce the retransmission of reusable multimedia objects over the Internet. Performance evaluation shows that the scheme reduces the jitter significantly while maintaining the needed QoS, and the scheme outperforms popular rendering formats.

Keywords: buffer, Internet, multimedia, movie, XML

1. Introduction

The demand for digital multimedia libraries and synthetic movies accessible over mobile devices is growing exponentially. People are using the Internet to retrieve archived media more than ever. The rate of increase in the demand for high quality rendering and transmission is much faster than the rate of increase of the transmission bandwidth in the Internet. With the new initiative of Mpeg-7 standards [6] cross-cultural exchange of multimedia objects will increase significantly, and will lead to real time collaborative multimedia modeling and information retrieval [8].

In order to maintain QoS in a video clip, at least 15 - 20 frames per second should be transmitted in addition to sufficient quality description of objects. Assuming a compressed data of 8KB/frame, a throughput of 160 KB/sec is needed to serve just one client. A simple sixty minute show will require around one GB of data transfer for just one client.

To reduce the excessive overhead for the repeated access of the same objects and to avoid hot spots server side push buffers [1, 9] have been used. Push buffer management uses a look-ahead scheme along with server side active buffer to interleave disk access and prefetching [10]. In order to restrain the bandwidth overload, client side pull buffers [1, 4, 10] have been used to cache and reuse the retrieved objects. However, little has been developed to coordinate the server side and client side buffers. This problem becomes severe to display movies and clips over mobile devices and PDAs which have limited processing and memory capabilities.

In our previous work, we had proposed and demonstrated a STMD ? <u>S</u>ingle <u>T</u>ransmission <u>M</u>ultiple <u>D</u>isplay ? model [3] to reduce bandwidth requirement for high resolution synthetic movies. STMD model extends MPEG-7's [6] object based representation by further decomposing a complex object as a hierarchical graph of reusable multimedia subcomponents, and the high quality realistic images are superimposed on the nodes and edges of the graphs. In STMD model, the multimedia subcomponents are transmitted once by a server, archived by the client, and retrieved at the client end when needed reducing the need for the retransmission of the subcomponent files comprising a complex media object. However, memory in mobile devices is restricted despite the recent increase. This constraint enforces deletion of media objects at client end in the case of buffer overflow resulting into retransmission of multimedia objects.

The goal of this research is to reduce jitters and maintain QoS by reducing the frequency of retransmission of previously transmitted subcomponents. In this paper, we describe a server directed buffer management scheme to display synthetic movie. The scheme is based upon the following criterion:

1) A movie is divided into multiple scenes. Each scene consists of multiple frames. Frames are represented using XML [11]. A server side static analysis is done

to analyze the frequency of reusable multimedia subcomponents in different scenes.

2) Based upon the static analysis of a movie and the look-ahead analysis at runtime, the multimedia objects are transmitted and archived in the client-end buffer for future retrievals.
3) The server keeps a local map of the archival state of each client and updates the corresponding local map after transmitting each frame. Based upon the analysis of the local map and the knowledge of memory capability of each client, and the available transmission bandwidth, the server directs each client to update their buffer. Based upon the direction of the server, a client retains or deletes an object from its buffer, or moves objects from active to passive buffer.
4) Client uses a back-channel (in a restricted manner) to inform the server about media objects lost during the original transmission.

The major advantages of the scheme are :

1) Static analysis of the movie optimizes the buffer management by identifying the exact memory requirement for future objects,
2) There is a coordination between the server and the client resulting into an optimum retransmission and retention of the reusable media subcomponents,
3) Criterion using the exact occurrence of objects in near future based upon static analysis and the knowledge of the size of the objects are used to decide the retention (and retransmission) of a reusable object in the client-end buffer.
4) The scheme is suitable for movies using MPEG-7 standards and schemes using XML based descriptions such as STMD model [3].

The major contributions of this paper are:

1) A static analysis of the object based movies using look-ahead windows has been proposed for better buffer management, and
2) A server directed scheme has been proposed for better coordination between server and clients for better memory management at the client end.

The scheme has been implemented as part of the STMDL (Single Transmission Multiple Display Language) project using Java, and supports object based streaming at the frame level.

The paper is organized as follows. Section 2 describes some background and new definitions needed for the static analysis of synthetic movies based upon STMD model. Section 3 describes the overall scheme. Section 4 explains the implementation at an abstract level. Section 5 describes the performance evaluation, and the last section concludes the paper.

2. Background

In this section, we briefly describe the previous approaches for buffer management, and introduce new definitions used in the static analysis.

2.1 Buffer management schemes

In any multimedia system, the delays are introduced during the disk access at the server end, during transmission caused by bandwidth saturation/ fluctuation, due to retransmission caused by packet loss, and during the rendering of objects. Bandwidth saturation is caused by excessive admissibility of the clients and/or variation of the client end demand resulting into hot spots. A multimedia stream consists of both large size still images and continuous media objects. A jitter occurs when the frame can not be rendered at the client end during the regular course of time.

In order to avoid seek time delays, in the past, many look-ahead schemes have been used for disk accesses [7, 9, 10]. The data is pushed by the server on the Internet in an optimized fashion to avoid the delay of data at the client end. A server performs analysis to identify the time to access and send the large size still-images or continuous media objects to reduce the delay caused by bandwidth variations.

In a movie being broadcasted on a frame basis, client side archiving after rendering a frame is needed if previously rendered media objects needs to be rendered again such as in STMD model or MPEG-7. In the absence of coordination from the server, client side discards incoming frames or discards archived frames without any knowledge of future data. The lack of coordination and the lack of look-ahead mechanism enforce retransmission of the same media objects multiple times. This bottleneck may cause unexpected jitters or result into compromise in the QoS if frames are duplicated (or reconstructed) to reduce the jitter.

Previous buffer management schemes have used standard MRU, LRU, and other algorithms [7] at the server end to predict the objects, and can be easily extended to manage the client end buffer to discard inactive objects. However, none of the techniques use static analysis of the synthetic movie to predict the buffer usage at the client end by future scenes.

2.2 Definitions

A movie is a sequence of scenes. A scene is a sequence of frames. Scenes are characterized by a common background or set of interacting objects.

Request time (T_L) is the time for which the client will wait before using the backchannel for requesting the server to retransmit a lost media object. *Transmission delay* (T_D) is the time duration for the server to receive a request for data retransmission from the client. *Rendering time* (T_R) is the total time taken by the client to render an object in the current frame. T_S is the seek time to transfer a media object from secondary memory to the server side buffer.

Unlike the notion of *'MRU'* that is based upon predicting future by analyzing the frames rendered in the past, we use the notion of *immediacy* – frames to be rendered in near future – based on exact static analysis of the entire set of consecutive frames to be rendered in future. Since we are interested in rendering archived synthetic movies, such a static analysis is possible.

A *look-ahead window* is a subsequence of frames (starting from the current frame) which are analyzed for the presence of active multimedia objects. An object is active if it occurs in multiple frames in a lookahead window or in the current scene. An object is *volatile* if the last occurrence of the object is only in the current frame being transmitted for rendering. An object is *persistent* if it occurs in more than one scene.

A client side active buffer contains active objects, and client side passive buffer contains passive objects – objects which do occur in future scenes yet missing in the current lookahead window and in the current scene.

Local frequency (denoted as *LF*) of an object is the number of frames in a scene in which the object has occurred. *Global frequency* (denoted as *GF*) of an object is the total number of frames in a movie in which the object has occurred. *Future local frequency* (denoted as *FLF*) of an object is the number of frames in a scene in which the object will occur after rendering the current frame. *Future global frequency* (*denoted as FGF*) of an object is the total number of frames in a movie in which an object will occur after the current frame. An *immediacy-count* (denoted by '*IC*') of an object is defined as the number of frames in the current look-ahead window containing the object such that those many frames can be transmitted without causing a jitter at the client end. *IC* is calculated based on the bandwidth and size of transmitted objects. The information about *FLF* and *IC* is needed to decide the active and volatile objects, and *FGF* is needed to retain the object in a passive buffer (secondary storage in mobile devices, if available) even after the objects are removed from the active buffer.

Buffering time (B_T) is the total time for which the client should have the data to continue rendering in case of transmission delay and network congestion. Buffering time B_T and B_S ? total buffer space at the client end ? are related by the equation $B_T = (B_S / b) * \textit{scaling factor}$ where '*b*' the bandwidth. The scaling factor depends upon the speed of rendering and the random variation of the transmission rate over the Internet. In the case of fast forward, the *scaling-factor* > 1 since the frames are being rendered at a faster rate needing more buffer space to reduce jitters. Similarly in the case of slow rendering the *scaling factor* < 1 since there is more time for transmission. For the Internet communication fluctuating randomly the *scaling factor* > 1 to accommodate the congestion over the Internet.

3. The overall scheme

In this section, we will describe the overall scheme of predictive buffer management at an abstract level.

The whole scheme contains a push buffer at the server end and a buffer at the client end. The server side map has two types of data structures for each client: an *object map* and a *scene-map*. An object map is a dynamic array of a 12-tuple < *object-id, frame-id list, object-type, size, LF, GF, FLF, FGF, IC, active / passive buffer bit, transmitted-bit, buffered-bit* >. A scene map is an array of triples <*scene-id, start-frame, end-frame*>.

The active/passive bit indicates the presence of the object in the active or passive buffer, the transmitted-bit is set to true after transmitting the object for the first time, and the buffered-bit is set to true after the media object is loaded from the secondary memory to the server side buffer, and turned to false after the object is deleted from the server buffer. This bit is necessary for the synchronization since there are two threads: main thread analyzes the object map, signals to start filling the server side buffer, and transmits the server side buffer; the second thread seeks the object from the secondary memory and fills up the server side buffer as explain in Section 4.

The server analyzes the object map to update the counts, identifies the objects to be deleted, and identifies the objects to be transferred from active to passive mode. The scene map along with the object map is used to reason about the hot spots where an object will be rendered frequently by the client, and should be maintained in an active buffer. Based upon the analysis of the object map and scene map, the server sends the buffer management instructions to client. Client upon the receipt of the analysis of the server takes appropriate buffer management actions.

After admitting a client, the server receives the information about the capability of the client based upon the message transmitted by the client during its request. Based upon this information, the server decides the buffer size, strategy to transfer the objects from active to passive buffer or to discard objects from the client side active/passive buffers.

The media stream from the server to the client consists of two types of information: *media* and *control*. Media information consists of video and audio files. *Control information* contains two types of frames: *object description frames* and *command frames*. *Object description frames* consists of XML description of the complex media objects using graph representations and object-id (for identification and synchronization) of the media objects. The *command frames* contain the commands to manage the client end buffer. The commands are to insert an object in the active buffer, delete an object from the active buffer, delete an object from the passive buffer, transfer an object from active to passive buffer or vica versa, and to increase the size of the allocated buffer

The server also manages its own server side buffer which is needed to reduce the seek time from the secondary storage such as disks for the repeated requests caused by lost packages during the transmission or to handle hot spots.

A static analysis of the complete synthetic movie is done to determine to initial *LF*, *GF*, *FLF*, *FGF*, and *IC* for every media object comprising the movie. Initial counts are stored in the frame before transmission, and object map is modified at runtime after transmitting each frame.

Before the movie starts, the server sends the size of the required buffer using an equation *Buffer-size = minimum(transmission rate ´ buffering Time ´ scaling-factor, allocable memory at the client's end*). The client allocates the buffer.

The initial object map is built after the analysis of first *L* frames comprising the initial look-ahead window. After analyzing first *L* frames, the look-ahead window is shifted by one frame at a time after transmitting the current frame (see Figure 1).

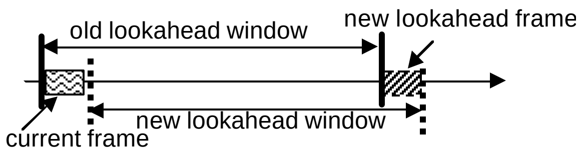

Figure 1: An overall lookahead scheme

After shifting the look-ahead window the counters in the server side object maps are updated. Based upon the command by the server, the media objects (archived at the client ends) are deleted, inserted, or transferred between passive and active buffer.

After transmitting the current frame, a buffer overflow is performed on the object map. Based upon the overflow check, the server issues commands to delete volatile objects, transfer least active objects to passive buffer, delete small objects with least FGF, and delete some of least active objects in favor of large still images to avoid data transfer overhead. All these commands are inserted in a command frame which is transmitted right after processing the current XML frame. These commands are executed by the client side manager after rendering the corresponding frame.

The information about each frame are transmitted in the order (Object-description frame, set of media objects in the frame previously not transmitted, command frame) for each frame. Set of media objects to be transmitted is decided after the analysis of the object map.

The process is repeated until all the frames have been transmitted to the client end.

3.1 Object map utilization and management

An *object-id* is needed for the reusability of an object to render objects in a frame and to access an object across multiple frames and scenes. The size of an object is needed to make a decision to discard the object in the presence of newly transmitted objects. If the size of an active/passive object is very large (such as still images), the object is preferentially retained over the smaller more active (with higher *FLF* and/or higher *IC*) objects to reduce the jitter caused by excessive data transfer overhead.

After the analysis and transmission of the current frame, the lookahead window is moved forward by one frame. The *FLF* and *IC* and *FGF* counters of all the objects in the transmitted frame are decremented by one. The *FLF* and *IC* counters of all objects occurring in the new look-ahead frame are incremented by one. An object with $FLF = 0$ and $IC = 0$ and $FGF = 0$ is a volatile object, and is immediately removed. The objects with $max(IC, FLF) > 0$ are retained as active objects. The notion of identifying *max(IC, FLF)* is useful for objects belonging to adjacent scenes near the scene boundaries. An object with *FLF* as *0* and *IC* as 0 and $FGF > 0$ is transferred from the client side active buffer and retained in the client side passive buffer unless there is not enough memory at the client end to retain the object. The information about the objects newly added in the lookahead frame (those objects which are absent in the object map) is transmitted from the server to the client. A passive object becomes active if it is present in the newly added look-ahead frame (IC = 1 or FLF = 1). An active object becomes passive after the shifting the look ahead window if $IC = 0$ and $FLF = 0$ and $FGF > 0$.

The total buffer requirement at the client end is given by the sum of the memory requirement of the active objects. If the sum of the memory requirement of the active objects is more than the buffer size, then depending upon the system configuration at the client end, one or more of the following actions are taken:

1) The server issues a command to increase the client end buffer size,
2) Some of the active objects are pushed from active to passive buffer if secondary memory is available at the client end, or
3) Some of the active objects are discarded since the cost to retransmit large size passive objects may be significant to cause jitter in future.

Following strategies are used to make a decision:

1) The active objects with the smallest *immediacy-counts* are pushed to a passive buffer (if available) to free up the active buffer space due to the lack of active buffer space,
2) Least persistent passive objects (decided by the smallest *FGF*) are discarded in the absence of available memory.
3) Among the objects with the same *immediacy-counts* or the same *FGF*, the objects to be discarded are selected to minimize the cumulative sum of seek time + transmission time upon re-request.

The overall abstract process is given in Figure 2. We describe an abstract description of the process code in pseudo-C like syntax. We denote a field of an object using an annotation of the form *object.field*.

```
Abstract process code: server_buffer_manager
Input: A movie XML file and a database of media objects;
         A look-ahead window size L;
Output: A stream of frames transmitted to a client;
{ set up a main channel $CH^M$ and a back channel $CH^B$;
   perform static analysis on movie XML file;
   buffer size $B^M$ = minimum(transmit_rate $\times$ buffering_time $\times$
   scaling, available memory in the client);
   create a client buffer of the size $B^M$, and a server buffer;
   create an empty server side object map $M^O$;
       /* look ahead analysis for the first L frames*/
   for (index =1; index $\leq$ L; index++){
      $S_1$ = set of object-tuples in current frame indexed by
      'index';
      for (every object-tuple O $\in$ $S_1$)
         if (O is absent in $M^O$){
            set initial O.LF and O.GF and O.IC from static analysis;
            O.FLF = O.LF; O.FGF = O.GF; o.active = true;
            O.transmitted = false; O.buffered = false;
            O.active = true;
            insert object tuple O in the object map $M^O$;
            }
         else increment O.FLF, O.FGF, O.IC by 1;
      }
   /* transmit and analyze the frames incrementally */
   for (index =1; index $\leq$ total-frames; index++){
      for (every request R in the backchannel $CH^B$)
           lookup $M^O$ and transmit objects with matching obj_id;
      create an empty 'Command frame';
      transmit the current object description frame indexed by
      index through $CH^M$;
      $S_{CF}$ = set of object-tuples in the current frame;
      start seek-and-buffer thread;
      for (every object-tuple O $\in$ $S_{CF}$) {
         /* process current frame */
         if (O.transmitted = false) {
            wait until O.buffered = true;
            transmit the object indexed by O.obj_id in $CH^M$;
            O.transmitted = true;}
      decrement O.FLF, O.FGF, O.IC by 1;
      if (O.FLF = = 0 && O.FGF = = 0 && O.IC = = 0) {
            insert command in the 'Command frame' to delete the
            object indexed by 'O.obj_id' from the active client
            buffer;
            delete the tuple O from $M^O$;}
      elseif (O.IC = = 0 && O.FLF = = 0 && O.FGF > 0){
            insert command in the 'Command frame' to move the
            object indexed by 'O.obj_id' from the active to the
            passive buffer;
            O.active = false;}
      }
    look_ahead_index = index + L;
    If (look_ahead_index $\leq$ total-frames) {
    $S_{NF}$ = set of objects in new frame;
    for (every object O $\in$ $S_{NF}$)
       /* process lookahead frame */
       if (O is absent in $M^O$ || O.active = = false in $M^O$){
            derive initial O.LF and O.GF and O.IC from static
            analysis;
            O.FLF = O.LF; O.FGF = O.GF; O.active = true;
            if (O is absent in $M^O$){
               O.transmitted = false; O.buffered = false;
                insert the tuple O in the object map $M^O$;}
            else insert command in the 'Command frame' to move
                   the object from passive to active buffer;
       else increment O.FLF, O.FGF, O.IC by 1;}
    commands = check_overflow(object-map);
    insert buffer update commands in the 'Command frame';
    transmit the 'Command-frame' through the channel $CH^M$;
     }
}
```

Figure 2. A server side coordinated management

3.2 Client side management

At the client end, the media frames, object description frames, and the command frames are separated. The object frames are analyzed and media objects are archived for rendering, and the command frame information is used to alter the client-buffer.

After receiving an object description frame, the references of the media objects in the frame are checked against the received media objects after (the sum of transmitted objects in the currently received frame containing the object) / b where b is the transmission bandwidth of the network. After determining the absence of an object, a client waits for T_L seconds before committing for the request for retransmission to avoid overloading the server and the Internet. Before sending a request the absence of the object is ascertained once more. For a jitter free rendering, the request time T_L is approximated by the equation $T_L < (B_T - T_R) -$ (Object-size / network bandwidth) $- T_S - T_D$. The first term $(B_T - T_R)$ gives the overall time taken to render the frames preceding the current frame being analyzed, the second term gives the time taken to transfer the data from the server to the client, T_S gives the data transfer time from the server disk to the server buffer, and the last term gives transmission delay to send a request from the client to the server. The request time for large size data objects is smaller since the data transfer time is larger. A value of $T_L < 0$ shows unavoidable jitter if the lost packet is requested. In such case, and the frame needs to be reconstructed using existing data if possible.

After rendering the objects in the frame the commands from the corresponding command frame are executed to update the active and passive buffers.

4. An implementation

In this section, we briefly explain an abstract implementation to manage server side buffer, update server side map, and client side buffer. The scheme was implemented using Java.

The management of the server side map and client side buffer is done concurrently after the server starts transmitting the frames to the client and before the transmission of L frames ($L > 0$) comprising the initial look-ahead window. The choice of L is decided by the overhead of managing such a table and savings in the overall data transmission.

After transmitting a frame, the object map is analyzed for potential overflow conditions by a *check_overflow*(*object-map*) routine. If the size of the client-end buffer is less than the cumulative sum of the archived media objects in the client buffer then the client buffer is cleaned up. While the analysis is done at the server end, the cleaning up is done by the client after rendering the corresponding frame based upon the command transmitted by the server. After the buffer cleanup, the current frame is discarded and the information from the next look-ahead frame is added. The process is repeated after transmitting every frame for rendering.

Both server side and client side have a circular queue of buffers. Each buffer stores a media object (or part of media object depending upon the size). The server side has two running threads: the first thread fills in the circular queue of buffers from the disk arrays, and the second thread transmits the buffers of media objects (in order) to the client. Both threads are synchronized so that first thread waits if the buffered media objects are not transmitted. The client side also has two threads: the first thread receives the data from the stream, and the second thread manages the client side active and passive buffers, composes the scenes, and renders the scene. We found that the use of two threads was sufficient for each client. The overall scheme is given in Figure 3.

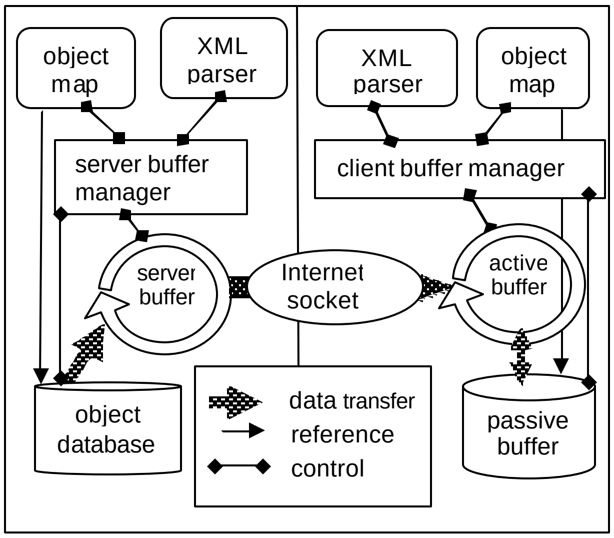

Figure 3. An implementation

5. Performance evaluation

We evaluated the system against *MRU* policy for the reuse of archived media objects, varied different size of the look-ahead window to measure the reduction in the overall data retransmission, and compared our scheme against popular systems such as Quicktime, extended AVI format, and MPEG-4 format.

A short movie of a walking human character with seven different nodes ? two legs, neck, two hands, head, and a torso ? was transmitted over the Internet. Overall 42 frames were transmitted, and the total object database had the size of 208 KB. The results are summarized in Table I and Figure 4.

Table I. Lookahead size vs. data transfer

Look-ahead Window	Data transfer in KB
0	1640
5	830
10	510
15	400
20	220

Table I shows that the increase in the size of the look-ahead window significantly reduces the overall data transfer. The look-ahead window size = 5 outperformed the savings in the data transmission using *MRU* policy, and the look-ahead window size of 20 gives an optimum savings in overall data transfer.

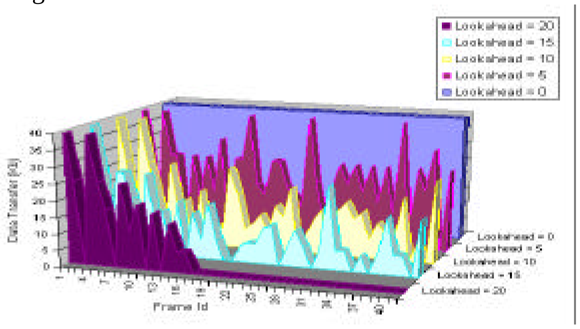

Figure 4. Data transfer vs. lookahead values

Figure 4 shows that the savings in data retransmission saturates after the look-ahead window size = 20 (the front graph in Figure 4). Our system needed less overall data transfer compared to all three formats for a look-ahead window size = 15

6. Conclusion

In this paper, we have integrated static analysis of synthetic movie with the server side coordination of the client buffer to improve the bandwidth requirement and QoS. The scheme uses look-ahead window and static analysis of the movie to reduce the retransmission of media subcomponents. The scheme is suitable for XML based models such as MPEG-7 and STMD — Single Transmission Multiple Display ? models since it can efficiently reuse the archived objects at the client end based upon server side analysis of the client buffer state. The buffer management scheme is suitable for mobile devices due to its capability of reducing retransmission overhead.

References

[1] S. Acharya, M. Franklin, Stanley, Balancing Push and Pull for Data Broadcast," *ACMSIGMOD 97*, 1997, pp. 183-194

[2] Y. Bai, and R. Ito, "User-Oriented Fair Buffer Management for MPEG Video Streams," *17th International Conference on Advanced Networking and Applications*, Xi'an, China, March 2003, pp. 241-247

[1] A. K. Bansal, T. Kapoor, and R. Pathak, "Extending XML for Graph Based Visualization of Complex Objects and Animation Over the Internet," *Proceedings of the Second International Conference on Internet Computing*, Las Vegas, June 2001, pp. 750-756

[2] M. Bhide, P. Deolasee, A. Kathkar, A. Panchbudehe, K. Ramamrtitham, and P. Shenoy, "Adaptive Push-pull: Disseminating Dynamic Web Data," *IEEE Transactions on Computers*, Vol 51, No. 6, June 2002

[3] S. Lee, H. W. Seung, and T. W. Jeon, "An Integrated Push/Pull Buffer Management Method in Multimedia Communication Environments," *LCTES 2000*, pp. 216-220

[4] J. M. Martinez, R. Koenen, F. Pereira, "MPEG-7 – The Generic Multimedia Content Description Standard, Part I," *IEEE Multimedia*, April-June Issue, 2002, pp. 78-87

[5] J. Nang and S. Heo, An Efficient Buffer Management Scheme for Multimedia File System," *IEICE Transaction of Inf. & Syst.*, Vol. E83-D, No. 6, 2000

[6] S. W. Ryan, A. K. Bansal, T. Kapoor, "A Distributed Multimedia Knowledge Based Environment for Modeling over the Internet," *Proceedings of the International Conference for Tools with Artificial Intelligence*, November 2000, pp. 140-146

[7] Y. Won and J. Srivastava, "SMDP: Minimizing Buffer Requirements for Continuous Media Servers," *Multimedia Systems*, Volume 8, Issue 2, pp.105-117, 2000

[8] W. Wright, "An Efficient Video-on-Demand Model," *IEEE Computer*, Volume 34, No. 5, pp. 64-70, 2001

[9] "Extensible Markup Language 1.0 (Second Edition)," *http://www.w3.org/TR/2000/* REC-xml-20001006

[10] "XML Parser for JAVA," *http://www.alphaworks.ibm.com /tech/xml4j*