Carnival - A low cost solution to Interactive Movie on Demand System

Sachin Agarwal Raman Mittal Aayush Deep Garg Sudip Sanyal
Indian Institute of Information Technology – Allahabad
Allahabad, India
{sachin, rmittal_02, aayush , ssanyal}@iiita.ac.in

Abstract

This paper aims at describing Carnival, a movie on demand system that provides Movie on demand service by implementing low cost storage architecture for movie storage server. Carnival uses personal computers with ordinary configuration, as movie storage servers. Since RAM will become a major bottleneck in such storage servers, hence optimization from the view of memory should be a primary concern. The solution described in this paper proposes a movie file system which minimizes the RAM requirements of the movie storage servers. The proposed file system provides various VCR functionalities like forward, rewind pause and play without any additional RAM requirements.

1. Introduction

1.1. Movie on Demand System

A movie on demand server is a system that store movies in compressed digital form and provides support for the movie data to be accessed by remote clients in real time. The Movie on demand service provides flexibilities to the viewer such that the viewer can start watching a movie from a collection stored on the server, at any time he/she wishes. The viewer can also use the VCR operations like pause, play, forward and rewind.

With increasing demand for large-scale Movie on demand systems, considerable effort has been spent in designing scalable, reliable, and cost-effective video servers. Nevertheless, a video server can only have finite capacity. As the system scales up, the server will need to be upgraded and this can become very expensive.

In our work (Carnival), personal computers with ordinary [1] configuration have been used as storage servers. RAM requirement is one of the major concerns which paralyses such movie servers. It follows an incremental trend with the increase in the number of connections. Since RAM is costly, it adds to the cost of the storage server. We have implemented a new file system that incorporates the RAM optimizations described in Fellini multimedia storage system [1]. The movie data can be interpreted as a 2D matrix with each cell corresponds to a block of the movie data. Carnival's file system stores this matrix in column major form on the disk. This storage strategy considerably reduces the RAM requirements while serving multiple clients for a single movie.

The implementation of the VCR functionalities further incorporates expensive overhead in RAM requirements. The available architectures [1, 12, 14] of the movie storage servers did provide some optimizations for the RAM requirements .In most of the cases, the use of VCR functionalities by the client adds unwanted overhead and/or deteriorates the performance of the server. We propose a method to implement the VCR functionalities like play, pause, fast-forward and rewind using this architecture, without any overhead of RAM or performance degradation.

1.2. Organization of the paper

The rest of the paper is organized as follows. Section 2 mentions some of the related works published in this field. Section 3 describes the details of different components of Carnival. Section 3.1 describes the architecture of storage server, including the storage and retrieval optimizations and the proposed method for implementation of VCR functionalities. Section 3.2 describes the file system architecture along with its detailed working. Finally we present our results and conclusion derived from simulation of the techniques described in this paper.

[1] CPU: Intel® Pentium® 4 CPU 1.5 GHz
RAM: 256 MBytes.

Proceedings of the 2005 International Conference on Computational Intelligence for Modelling, Control and Automation, and International Confe
Intelligent Agents, Web Technologies and Internet Commerce (CIMCA-IAWTIC'05)
0-7695-2504-0/05 $20.00 © 2005 IEEE

2. Related Work

Current Movie on Demand systems can be classified into centralized [2, 3] and distributed [4, 5] architectures.

Serpanos, *et al*. [3] compared the performance of centralized and distributed architectures for video servers. Their work concluded that in general a centralized architecture is gives good performance and management but it is not preferable due to high cost and scalability issues.

Researches have contributed a lot of their efforts in the development of multimedia file systems optimized for movie on demand services. Rangan *et al.* [6] proposes a multimedia file system for continuous storage and retrieval of media and also discusses the admission control algorithms. Freedman *et al.* [7] discusses a scalable file system for multiple disks for the intensive I/O applications and multimedia systems. The Tiger video file server [8] implements a distributed file system that strips the files over multiple disks and the computers in distributed fashion.

The Fellini[1] continuous Multimedia storage system by AT&T Bell Labs has described a storage technique for efficient retrieval of the movie data to minimize the RAM requirements.

VCR functionalities too are the major issues in a movie on demand system. Qazzaz *et al.* [9] describes allocating some resource channels for providing VCR functionalities. It uses large size buffers at client side. It continues to transfer multicast data to the client, even if he comes late or use pause etc. and uses different unicast channel to handle VCR operations. Venkatram *et al.* [10] propose segmentation of multimedia data into small segments and segment-skip for VCR operations. Law *et al.* [11] also propose a possible implementation of VCR functionality.

3. Detailed description of the System

3.1. The Storage server

3.1.1. Movie Storage. The movie is stored on a raw partition/ hard disk using our proposed file system (described in section 3.2). The file system is mounted on the raw partition/ hard disk to store the movies. The movie could be interpreted as two-dimensional matrix [12] i.e. movie data is divided into blocks of equal size where each block corresponds to a cell in the two-dimensional matrix. The proposed file system stores the movie data in the column major form on the disk.

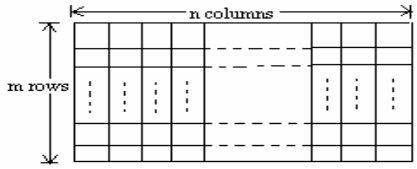

Figure 1. The 2D matrix used to store the movie data

For example, suppose a movie is interpreted in the form of a 2D matrix of dimension 4 rows and 3 columns then the sequence of blocks stored on the disk would be : 1,5,9,2,6,10,3,7,11,4,8,12.

3.1.2. Optimization during retrieval of movie data. To retrieve any data block of the movie, corresponding column is loaded in the memory [12]. In order to serve the movie stream sequentially to the client, the first column (comprising of the blocks numbered 1, 5 and 9 in the example given in section 3.1.1)) needs to be loaded into the memory. While this column is being served to the client, the second column, consisting of the blocks numbered 2, 6, and 10, is fetched from the disk. The columns are sequentially retrieved starting from the first column to the last and then wrapping back to the first column. In order to serve the same movie to multiple clients, instead of retrieving multiple blocks of movie for each client, only a single column would serve any number of clients. Considerable RAM optimization is achieved when the number of clients is much more than the number of blocks in a single column of the 2D matrix. The subsequent columns are stored sequentially on the hard disk, hence their sequential retrieval ensures the minimization of disk head seek time and rotational latency. The position in a movie stream from where the data is transferred to a particular client is called a phase. The number of unique phases supported will be equal to number of rows in 2D matrix. A two column length buffer can be used, one to retrieve the data and the other to transfer the movie data to the client in parallel. This technique effectively reduces the delay on the client side.

3.1.2. Implementation of VCR functionalities. We propose a novel method to implement the VCR functionalities using the above storage and retrieval technique. The various VCR functionalities like play, pause, fast-forward and rewind can be provided to the client, without any overhead of RAM or performance degradation. They have been described hereunder.

(a) Pause: To handle the pause functionality a buffer of size equal to the size of a single row of the two-dimensional matrix described in section 3.1.1), is maintained on the client side. When the client issues a pause command, the playback of the movie is stopped but the transfer of movie from the server is paused only

after the buffer on the client side gets exhausted. On un-pausing, the playback is started from the buffer. The data transfer from the server is started when the column containing the next data block required by the client is loaded into the memory and subsequent data is appended to the client's buffer.

(b) Fast Forward: To provide the fast forward functionality, instead of sending the next data block of the sequence, the data block present diagonally below in the two-dimensional matrix is send. The sequence of the data blocks sent will be **i, i+(n+1), i+2*(n+1)** and so on (where n is the number of columns in the two-dimensional matrix).

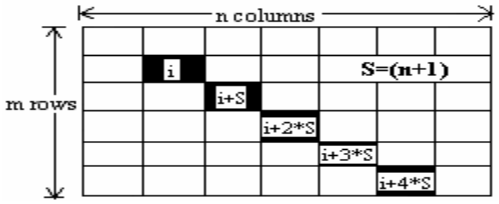

Figure 1. Blocks sent during Fast-Forward starting from i^{th} block

(c) Variable speed fast forward: To provide fast forward functionality at variable speeds the above described scheme can be modified as follows. Instead of sending a single block of data to the client to the client from a column, the block immediately below the block to be sent is also sent to the client. The frames corresponding to this block are displayed before the frame corresponding to the block diagonally below. This way the speed of fast forward on the client side is halved. So the sequence of blocks being displayed at client will be **i, i+S-1, i+S, i+2S-1, i+2S, ..** and so on. A buffer to hold at most **n** data blocks will be required at client. The speed of fast forwarding can be further reduced by increasing the no. of extra data blocks being sent to the client from each column. If **k** data blocks are sent from each column, then at client side before displaying the diagonal block the frames corresponding to last **k-1** data blocks in the same row which are already buffered on the client will be displayed. So a **k** time reduction in fast forward speed can be achieved. The sequence of blocks being displayed at client will be **i, i+S-(k-1),i+S-(k-2),…i+S-1, i+S, i+2S-(k-1), i+2S-(k-2),…i+2S-1, i+2S, ..** and so on. A buffer to hold at most **k*n** data blocks will be required at client.

(d) Rewind: In case of rewind command by the client, the next data block to be send is the data block present diagonally upward from present block in the two-dimensional matrix. The sequence of the data blocks sent will be **i, i-(n+1), i-2*(n+1)** and so on.

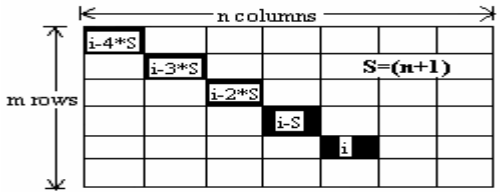

Figure 2. Blocks sent during Rewind starting from ith block

(e) Variable speed Rewind: Rewind functionality can be modified in a similar manner as fast forward to provide variable speeds of rewind. Only instead of sending **k-1** blocks below a designated block in a column, **k-1** blocks above it are sent. The sequence of blocks displayed at client will be **i, i-S+(k-1),i-S+(k-2),…i-S+1, i-S, i-2S+(k-1), i-2S+(k-2), ….., i-2S+1, i-2S, ..** and so on.

3.2. The File system

3.2.1. Architecture of the multimedia File system. The file system on the Carnival's movie server accounts for efficient retrieval and storage of the movie files so as to minimize the disk head movement and the memory requirements in case of multiple access by various clients or otherwise. The files and directories on the disk are represented internally in the form of inodes. A movie file is represented in form of a movie file inode and a directory is represented as a directory inode. An inode keeps general information about the file or directory such as its name, file number, access permissions, last modified times, file owner's id, group number, datazones and other required relevant details.

Carnival's movie specific file system is developed specifically for the movie on demand server. The file system has the following layout:

1. *Boot block* – consists of the appropriate information required to boot the operating system.
2. *Super block* –
   (a) Number of blocks for File Inode Bitmap = 1
   (b) Number of blocks used for Directory Inode Bitmap =1
   (c) Number of blocks for file zone Bitmap = 1
   (d) Number of blocks used for directory zone Bitmap = 1
   (e) Number of direct zones in the inode structure= 10
   (A single movie file on Carnival's file system can have a maximum size of 1.725 GB which is sufficient amount of space required by a single movie file.)

Number of free inodes and zones for directories and files are computed by referring the corresponding bitmap and the field is set accordingly.

3. *Directory inode bitmap* – It keeps information related to directory inodes.
4. *Movie File inode bitmap* – It keeps the information about the filled and empty movie file inode structures on the disk in form of 0's and 1's where 1 represents a filled inode and a 0 represents an empty inode.
5. *Directory zone bitmap*- It keeps the information about the filled and empty directory datazones using 0's for free and 1's for filled zones.
6. *Movie file zone bitmap* – It keeps the information about the filled and empty movie file datazones in the manner similar to the inode bitmap in form of 0's and 1's.
7. *Directory inodes* –It consists of the series of directory inodes structures present on the file system. Its structure is same as that of the Movie File inode structure described below.
8. *Movie File inodes* – The inode structure(in implementation) consists of the following fields:

(a) The size of the file in bytes.
(b) An array containing the zone numbers of the file data. The maximum size of this array is equal to the total number of direct zones.
(c) The device number of the device on which the file is stored.
(d) The inode number of this inode on the disk.
(e) The inode type, specifying whether the inode is a movie file inode or a directory inode.
(f) The count specifying the number of processes currently using the file.
(g) The total number of direct zones for this inode.
(h) A pointer containing the address of the super block.
(i) A field that marks whether any of the inode data has changed. In case any of the inode field undergoes a change, the field is set to dirty and the inode is written on to the disk from the inode cache later on.

9. *Directory content Blocks* – It holds all the data of directories. These directory content blocks reside within a group of four blocks on the file system. The size of directory content block is taken to be 1024 bytes. Two directory content blocks constitute a single directory datazone for storing the data of any directory. On any standard file system the block size used for storing the file's data or the directory's data is same. But in Carnival's movie specific file system, the block size is 8.03MB (explained below), which is too large just for storing the content of any directory on the file system. So in order to prevent the wastage of disk space, the block size used for directory content has been reduced to 1024 bytes.

10. Movie Data Blocks –Since the movie data is large in comparison to the files normally stored in a typical file system, the block size taken in our case is quite large (235* size of a cell of the 2-D matrix). Here 235 is the maximum number of parallel phases supported by the movie server for a particular movie as described in the section 3.2.2) above. This has been calculated using the relation [12].

$$p= (d/r_d – a_t)*r_t/d \quad (1)$$

where
d = size of a cell of the 2-D matrix (here after referred as CELL_SIZE).
It has been calculated taking in account the size of main memory and the maximum number of different movies that can be transmitted simultaneously from the server. On a server having 256MB RAM to support 15 different movie streams simultaneously, size of a cell of the 2-D matrix is taken to be 35KB.
So the block size = 235*35KB ~ 8.03MB

r_d = rate of transfer of MPEG movie =1.5 Mbps
a_t = access time taken by the head of the hard disk
r_t = transfer rate of the hard disk.

On our hard disk configuration we obtained:-

a_t = 13.6
r_t = 400 Mbps

Hence p was calculated as 235 which is the number of rows of the 2-D matrix. The number of columns (n) of the matrix can be calculated according to the following relation:
n=sizeof(movie)/(p*d) (2)
In our case for a 700MB MPEG movie file, n was calculated out to be 88.

Each movie file datazone in Carnival's file system contains 22 blocks (BLOCKS_PER_ZONE for movie). This is chosen by assuming the average length of a MPEG movie file to be 700MB and block size equal to 8.03MBytes. Inodes on our (Carnival's) movie specific file system does not have any reference to indirect

zones which is present in many current file-systems. This is done intentionally so as to reduce the overhead associated with referencing of indirect zones for movie file data. As a consequence of large block size on the file system a very limited number of zones can do the job of storage of an average 700MB MPEG movie file.

3.2.2. Working of the File System.

(a) Initialization of the file-system

To initialize the file system, the super block is loaded from the hard disk into the file buffer cache. Various file system specific attributes are derived from the super block. A pointer which holds information about the current directory is made to point towards the root inode whose address is derived from the super block.

(b) The Buffer Caches and the Inode Caches

The buffer cache is introduced to minimize the disk operation. In Carnival's movie specific file system we have developed two buffer caches each for file and directory content blocks. Each buffer cache is maintained in form of a Hash Map with Hashing function as (HASH_MASK & Block number) where '&' is the logical AND operator.

A free list of buffers is also maintained that preserves the least recently used order. The free list is a doubly linked circular list of buffers with a dummy buffer header which marks its beginning and end. Whenever a new buffer has to be allocated, free list is searched and the returned buffer is inserted at appropriate index in the HASH MAP.

In Carnival's file system: The size of file Buffer cache is equal to 40 and HASH_MASK is 63. The size of directory Buffer cache is equal to 10 and HASH_MASK is 31. The layout of the buffer cache can be depicted from the figure below.

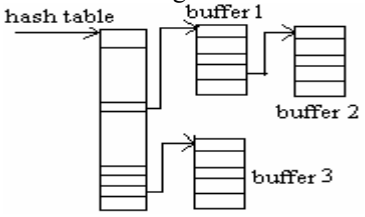

Figure 4. Structure of a single Buffer cache

Both inode caches (directory as well as file inode cache) are simply a continuous array of inode structures in the memory. When the I/O operation for an inode is to be done, the inode structure is first searched in the respective inode cache. If cache hit occurs then the inode structure need not be read from the disk and thus saves valuable time. In Carnival's file system the size of file inode cache is equal to 64 and that of directory inode cache is 32.

(c) Writing a new movie file to the file system

As described in section 3.1.1 the movie data is stored in a column major form in the file system. Whenever a new movie is to be stored on the disk, the complete target path of the movie file destination is provided to the file system. The given path is resolved and the directory in which the movie file is to be written is returned using to the algorithm given below.

```
Algorithm directoryi
Input: path name
Output: inode of target directory
if (path name starts from root)
        working inode=root inode (algorithm iget[19]);
else
        working inode=current directory inode (algorithm
        iget [19]);
read next path name component from input;
while (there is more path name)
    read directory (working inode) by use of algorithms
    bmap [19],bread[19],brelse [19]
    if(component matches an entry in directory (working
inode))
         get inode number for matched component;
         release working inode (algorithm iput);
         working inode = inode of matched component
        (algorithm iget);
    else
        create entry of component in working inode;
    read next path name component from input;
end while
return (working inode);
```

For example:-

Suppose the path given to the file system is /A/B/C/D.mpeg. The directory A already exists in the root directory of the file system but there are no directories named B and C on the file system. The above algorithm creates directory B in directory A and C in directory B and returns the working inode which is the directory inode of C. Now the movie file inode bitmap is scanned for the first available free bit, the bit is set to unity and the corresponding inode number is returned. This inode number can now be used to uniquely identify the file on the hard disk.

The storage of the movie in column major form is done by reading appropriate chunks of CELL_SIZE of movie data from the movie source. When the size of the buffer becomes equal to the block size of the file system, it has to be written onto the disk. The offset in the movie source file to read the appropriate chunks of movie data can be calculated according to the equations given below.

for each block present in i^{th} column and j^{th} row

$$\text{Offset} = (i + j * (\text{no of cols})) * \text{CELL\_SIZE} \quad (3)$$

The data blocks of the movie inode are organized by grouping them into various zones with maximum

number of blocks in each zone equal to BLOCKS_PER_ZONE (for movie) of the file system.

A zone is created by scanning the movie file zone bitmap and finding the first available zone and making the entry of the corresponding zone number in the movie inode. The inode cache is searched for the availability of the inode and all the additions and updations to the inode fields are temporarily done to this alias of the disk inode in the inode cache. This leads to the setting of DIRTY field in the inode structure indicating that its fields has been changed and it needs to be written back to the disk with its new fields. This writing of inode structure on the disk is done only when this inode structure is not in use and a new inode structure needs to be assigned in the inode cache for some other file.

(d) Reading a movie file from the file system

The movie data is written in the column major form to the file system and is retrieved in the same form during its read from the file system. To read the data contained in a movie file, its inode is loaded into the RAM by resolving the given movie file path name using the *namei[13]* algorithm. While the file is being read from the hard disk, one complete column is read at a time. The column to be retrieved is based on the next data block to be send to the client and is known before the read operation. The movie file data block (DB) on the file system containing the movie file data corresponding to this column can be calculated from the following relation:

DB = SOD + (DZ*BPZ)+ Column no. (4)

Where

(i) SOD is the number of blocks to be skipped on hard disk in Carnival's file system before reading any data blocks. For reading the data block of any movie file SOD includes the skipping of the boot block, super block, all the bitmaps, directory inode and movie file inode structures and four blocks which contains all the directory content blocks.

(ii) DZ is the starting zone number for the inode of the file for which the data is requested. This starting zone number can directly be fetched from the field of the inode structure with is an array containing the zone numbers of the file data.

(iii) BPZ=BLOCKS_PER_ZONE (for movie)

NOTE: In Carnival's case the *bmap [13],bread [13] and brelse [13]* algorithms used i*n namei[13]* algorithm operates on the directory blocks. b*map [13]* operates on directory inodes, *bread [13]* and *brelse [13]* operates on directory content blocks.

The required data block calculated above is fetched by firstly checking for its availability in the file buffer cache. A cache hit saves the time required to read the data from the disk which is quite expensive. If it is not found in the file buffer cache, the respective block is read from the hard disk into the file buffer cache. Now the location of the block in the file buffer cache is copied into a temporary buffer which is then used to transfer the movie data to the client. To further lower the hard disk head movement the C-Scan [14] algorithm is used to service the multiple clients read requests.

4. Results

To get a rough estimate of the performance of Carnival, we ran some very simple experiments, in which a client program downloads a file from the storage server and the server sends the file by retrieving the movie file as described above. The whole experiment was carried out on a heavy traffic 100Mbps Ethernet Local Area Network. The plot of Transfer time versus Size of the movie has been shown below.

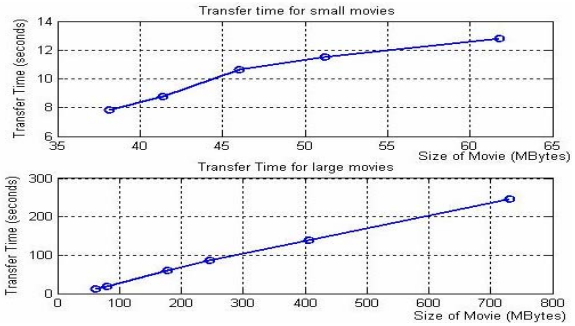

Figure 5. Plot of Transfer time against size of the movie

The first graph shows the transfer time for small (varying from 35 MBytes – 62 MBytes) sized movies. The graph clearly shows that the slope of the plot is increasing with increase in size of the movie due to increase in page faults (here page fault refers to not being able to find the requested block in buffer cache). Since the number of columns in the 2D matrix representation of a movie file are served in a round robin fashion, therefore for small sized movies, the subsequent columns might still remain in memory (buffer cache of the file system) and they need not be fetched from the memory, thus decreasing the number of disk accesses and hence the transfer time. The second graph shows the Transfer time for movies with larger (70 MBytes – 720 MBytes) size. Here the slope of the plot is almost constant because the number of page faults (not being able to find block in buffer cache) becomes constant because each new column has to be fetched from the hard disk. The plot of free memory versus number of clients for the same movie on the movie server has been shown below.

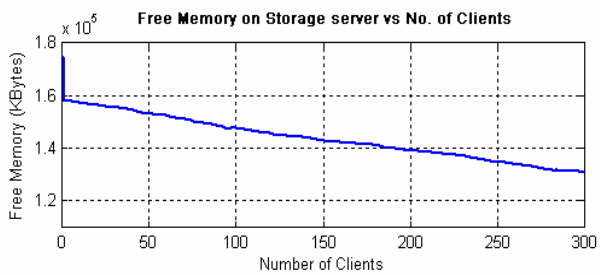

Figure 6. Free Memory available against the number of clients watching the same movie from the same server

The initial drop in the graph shows a memory usage of approximately 16 MBytes (since two buffers of size 8.03 MBytes each as explained in section 3.2.1), are loaded into memory). The initial drop in free memory will come as soon as the 1st client asks for the movie. Thereafter, there is no considerable increase in memory usage with the increase in number of clients. This is true even in the case of hundreds of clients as depicted by the graph above.

5. Conclusion

The storage optimization technique described in this paper along with complete implementation details of the file system and network architecture provides comprehensive details of Carnival, our movie on demand system. The storage enables the different sections of a movie to be concurrently retrieved from the disk and a number of clients could be simultaneously served thus minimizing the RAM requirements and thus the overall cost of the storage server. The proposed solution for VCR functionalities provides an efficient method for their implementation without any overhead of RAM requirements. The results show considerable optimizations in main memory (RAM) usage when hundreds of clients watch the same movie from the same storage server.

7. References

[1]. C. Martin, P. S. Narayan, B. Ozden, R. Rastogi, and A. Silberschatz. "The Fellini Multimedia Storage Server. Multimedia Information Storage and Management", Editor S. M. Chung, Kluwer Academic Publishers, 1996.

[2]. A. Krikelis, "Scalable multimedia servers", *IEEE Concurrency*, vol.6(4), Oct.-Dec. 1998, pp.8_10.

[3]. D.N. Serpanos, A. Bouloutas, "Centralized versus distributed multimedia servers", *IEEE Transactions on Circuits and Systems for Video Technology*, vol.10(8), Dec. 2000,pp.1438_1449.

[4]. F. Schaffa and J.-P. Nussbaumer, "On bandwidth and storage tradeoffs in multimedia distribution networks", in *Proc. Infocom*, 1995, pp.1020–1026.

[5]. L.A. Rowe, D.A. Berger, and .E. Baldeschwieler, "The Berkeley Distributed Video-on-Demand System, Multimedia Computing", *Proc. Sixth NEC Research Symp.*, T. Ishiguro, ed., Society for Industrial and Applied Mathematics, Philadelphia.

[6]. P.Venkat Rangan , Harrick M. Vin, "Designing file systems for digital video and audio", *Proceedings of the thirteenth ACM symposium on Operating systems principles*, p.81-94, October 13-16, 1991, Pacific Grove, California, United States

[7]. Craig S. Freedman , Josef Burger , David J. DeWitt, "SPIFFI-A Scalable Parallel File System for the Intel Paragon", *IEEE Transactions on Parallel and Distributed Systems*, v.7 n.11, p.1185-1200, Nov 1996.

[8]. Bolosky96 W. Bolosky, J. Ban'era III, R. Draves, R. Fitzgerald, G. Gibson, M. Jones, S. Levi, N. Myhrvold, R, Rashid, "The Tiger Video Fileserver". In proceedings of the *Sixth International Workshop on Network and Operating System Support for Digital Audio and Video (NOSSDAV 96)*, April, 1996

[9]. B.Qazzaz, R. Suppi, F. Cores, A. Ripoll, P. Hernandez, E. Luque. "Providing Interactive Video on Demand Services in Distributed Architecture". In Proceedings of the *29th EUROMICRO Conference "New Waves in System Architecture" (EUROMICRO'03),* page 215, Year 2003.

[10]. P.Venkatram, Shashikant Chaudhary, R. Rajavelsamy, T. R. Ramamohan, H. Ramakrishna,"Disk-oriented VCR operations for a multi-user VOD system", *Journal of Indian Institute of Science*, Sept.–Oct. 2004, ,Pg 123–140

[11]. Kelvin K. W. Law, John C. S. Lui, Leana Golubchik "Efficient support for interactive service in multi-resolution VOD system", *The VLDB Journal — The International Journal on Very Large Data Bases archive* Volume 8, Issue 2 (Oct 1999)Pages: 133 - 153

[12]. Banu Özden , Alexandros Biliris , Rajeev Rastogi , Abraham Silberschatz, " A Low-Cost Storage Server for Movie on Demand Databases", *Proceedings of the 20th International Conference on Very Large Data Bases*, p.594-605, September 12-15, 1994

[13]. Maurice J. Bach, *The Design of the UNIX Operating System*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1986

[14]. A.L. N. Reddy and J. C. Wyllie, *I/O issues in a multimedia system.* Computer, 27(3):69--74, March 1994.

[15]. Rowe, L. A., Patel, K. D., Smith, B. C., And Liu, K. "MPEG video in software: Representation, transmission, and playback", *In High Speed Network and Multimedia. Computing, Symp. on Elec. Imaging Sci. & Tech.* (San Jose, CA), Feb. 1994.