An Efficient Storage Mechanism to Distribute Disk Load in a VoD Server

D.N. Sujatha[1], K. Girish[1], K.R. Venugopal[1], and L.M. Patnaik[2]

[1] Department of Computer Science and Engineering
University Visvesvaraya College of Engineering Bangalore University
Bangalore-560001, India
suj_sat@yahoo.com
[2] Microprocessor Applications Laboratory, Indian Institute of Science
Bangalore-560012, India

Abstract. In this paper, a storage mechanism is devised to balance the load and to provide immediate service to the clients with a start-up delay of 2ms to 7 ms. The video storage is based on the probability of the clients requesting for the video. Videos with higher probability of being requested are stored and replicated to ensure guaranteed retrieval. Parity generation scheme is employed to provide reliability to non-popular videos. The system is also capable of handling disk failures transparently and thereby providing a reliable service to the clients.

Keywords: Fault Tolerance, Load Balancing, Start-up Delay, Video Server.

1 Introduction

Recent developments in storage mechanisms are making high performance Video-on-Demand (VoD) servers a reality. The video server stores heterogeneous information on array of high capacity storage servers and deliver them to the geographically distributed clients. The design constraint is to develop a large-scale cost-effective video server with a scalability to admit and service the client's requests simultaneously.

2 Previous Works

A comparison of different RAID levels is made to bring out Random Duplicate Assignment (RDA) [1]. In this strategy the video is striped and instead of being stored sequentially they are randomly allocated in different disks and each strip is mirrored to enable fault tolerance. The time required to access next block in the disk increase as the blocks are randomly allocated. A map between the blocks is to be maintained to handle this problem, which is an additional overhead. Golubchik et al. [2] discusses fundamental issues associated with providing fault tolerance in multi-disk VoD servers. In [3], [4] Replication of videos and placement of video blocks based on popularity is discussed.

3   System Architecture

The overview of storage mechanism is shown in Fig. 1. The video is divided into blocks based on number of disks. Each block is stored in different disks sequentially so that only once a block of video stored on the disk. The first disk to store the video is rotated to ensure that the load is balanced among the disks. If the block requested is stored in a disk, which is serving another client, the request is queued. The requests in the request queue are serviced in round robin fashion. If the video is popular then, the video is stripped across the array of the disks and mirrored. The request is serviced by the mirrored storage, in case of increase load for the popular videos. The mirrored data is also accessed in case of disk failure containing the popular videos. If the video is non-popular video, it is stripped across the disks with the last disk in the set of disks to store the parity information. Performing XoR operation between all the blocks of data generates parity block. This helps to rebuild the video block in case of disk failure.
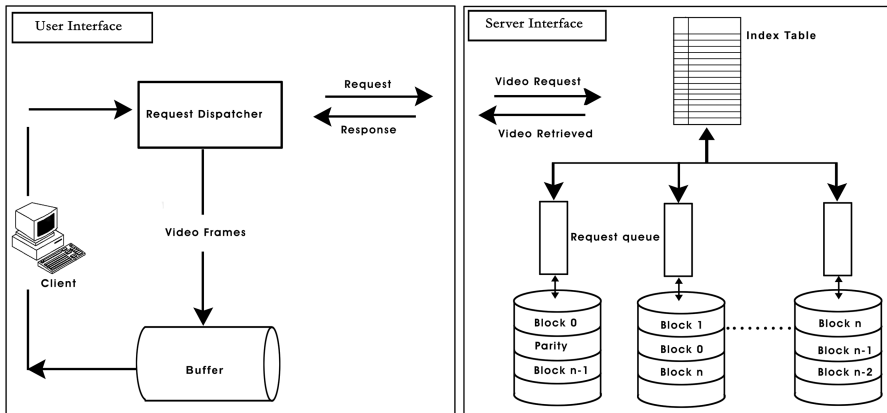

Fig. 1. Overview of the Storage Mechanism

4   Algorithm

A: Video_Storage (VideoId)
1. Determine the popularity of the video.
2. If (video is popular) *then* BlockSize = VideoSize / n
                   else BlockSize = VideoSize / n-1
3. If(space available to store video at BlockSize on disk j)
   $While(VideoSize \geq 0)$
   begin
           $j = ((VideoId\%n) + i)\%n$
           $reduceVideoSizebyBlockSize.$
           $storeblockondisk$ j

$\qquad$ *increment i by* 1
end
4. $if(video is popular)$ $\quad$ *then mirror the blocks in mirror storage.*
$\qquad\qquad\qquad$ *else* $\quad P_j = B_0 \oplus B_1$

$\quad$ While $(i < n)$
$\quad$ **begin**
$\qquad$ $p_j = p_j \oplus B_i$
$\qquad$ Store P_j in disk j
$\quad$ **end**

B: Handle_Request (Video_id)
1. found = search index table for VideoId
2. While (found=EOF)
3. Retrieve video_info from index table.
$\quad$ **begin**
$\qquad$ if (block not corrupted) and if (disk not loaded)
$\qquad\quad$ **begin**
$\qquad\qquad$ stream block i from disk j
$\qquad\qquad$ j = j+1
$\qquad\quad$ **end**
$\qquad$ *else* if (video is popular) then handle request from replica disk
$\qquad\qquad$ *else* Forward_Request (Video_Id)
$\qquad\qquad\quad$ *else* if (video is not popular)then rebuild block from parity block
$\qquad\qquad\qquad$ *else* stream from replica.
$\quad$ **end**

The storage routine shows how the videos are stored in the server to facilitate load balancing and the HandleRequest routine is designed to illustrate the behavior of the server on arrival of the request.

4.1   Illustrated Example

Consider the storage of 5 videos V_0, V_1, V_2, V_3, V_4 of file size 1500Mb, 1000Mb, 2000Mb, 800 Mb, and 3000 Mb respectively, where V_0, V_1, V_4 are popular videos and V_2, V_3 are non-popular videos. The disk-id of first block is disk-1, disk-2, disk-3, disk-4 and disk-5 with block size 300 Mb, 200 Mb, 800 Mb, 169 Mb and 600 Mb respectively. Considering 10 disks in the video server with 5 primary disks, the storage allocation for each video in their corresponding disks is given in Table 1. The request arrivals are indicated in Table. 2. The requests are served with each request being allocated bandwidth (Refer Table. 3). If the disk is busy serving different client it is moved to the request queue (Refer Table. 4). The request queue is checked for every 5 sec and if queue is not empty then, the clients are served in round robin fashion. It is assumed that the request queue capacity is 5. If the number of requests increases or the delay increases more than 10 ms then, the request is forwarded to other video server.

Table 1. Storage of Videos

Video-id	Block-size(Mb)	Disk-1	Disk-2	Disk-3	Disk-4	Disk-5
V_0	1500/5=300	B_0	B_1	B_2	B_3	B_4
V_1	1000/5=200	B_4	B_0	B_1	B_2	B_3
V_2	2000/4=500	B_3	P	B_0	B_1	B_2
V_3	800/5=160	B_2	B_3	P	B_0	B_1
V_4	3000/5=600	B_1	B_2	B_3	B_4	B_0

Table 2. Request Arrival

Request	Request-id	Arrival clock time(secs)
R_0	3	0
R_1	1	5
R_2	1	5
R_3	3	5
R_4	0	8
R_5	1	8
R_6	2	15
R_7	0	15
R_8	4	15
R_9	0	18

Table 3. Bandwidth Allocated to Requests

Time(ms)	Disk-1	Disk-2	Disk-3	Disk-4	Disk-5
T0=00	R_0(600Mb)	2Mbps			
T1=05	R_1(200Mb)	R_0(590Mb)	1Mbps	1Mbps	
T2=08	R_4(300Mb)	R_1(197Mb)	R_0(587Mb)	0.6Mbps	0.6Mbps
T3=10	R_4(298.2Mb)	R_0(200Mb)	R_3(160Mb)	0.6Mbps	0.6Mbps
T4=15	R_4(295.2Mb)	R_5(200Mb)	R_6(800Mb)	R_0(585.2Mb)	R_8(600Mb)
T5=18	R_4(294.2Mb)	R_5(200Mb)	R_6(499.4Mb)	R_0(584.6Mb)	R_8(599.4Mb)
T6=20	R_7(300Mb)	R_1(195.2Mb)	R_6(499Mb)	R_3(157Mb)	R_8(599.9Mb)

Table 4. Request Waiting in the Request Queue

Time(ms)	Disk-1	Disk-2	Disk-3	Disk-4	Disk-5
T0=00					
T1=05	R_2(200Mb)	R_3(160Mb)			
T2=08	R_2(200Mb)	R_3(160Mb)	R_5(200Mb)		
T3=10	R_2(200Mb)	R_3(585.2Mb)	R_1(195.2Mb)		
T4=15	R_7(300Mb)	R_1(195.2Mb)	R_3(157Mb)	R_2(197Mb)	
T5=18	R_7(300Mb)	R_7(197Mb)	R_0(584.2Mb)	R_9(300Mb)	R_2(199.7Mb)
T6=20	R_9(300Mb)	R_2(197Mb)	R_0(584.2Mb)	R_4(294.2Mb)	R_5(199Mb)

5   Simulation and Performance Analysis

Fig. 2, illustrates the start-up delay of the clients to begin downloading the video
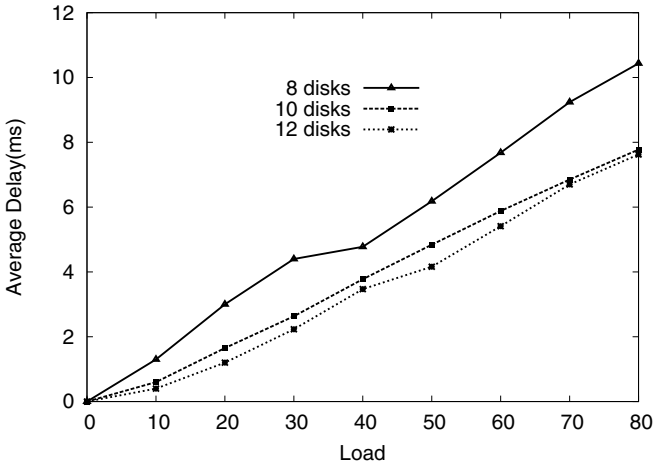after the request is made. The average delay increases with the increase of load, as
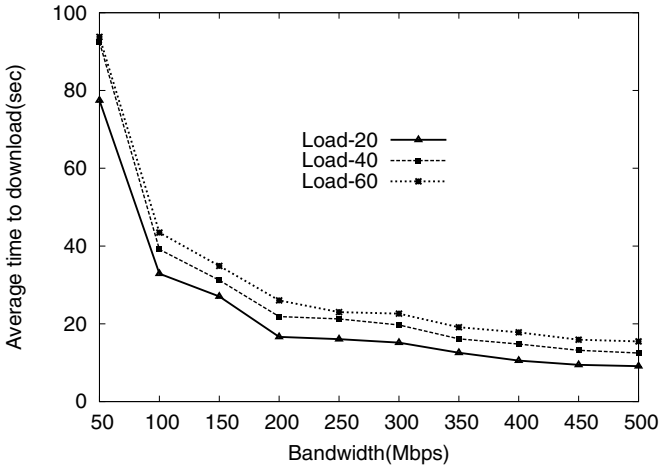
Fig. 2. Start-up Delay

Fig. 3. Bandwidth Utilization for Varying Load

the clients are queued if the disk is busy serving other clients. The delay increases by 17% with the increase of load. The start up delay can be further decreased with increase in number of disks in the system which is evident from the graph. Fig. 2, also depicts the decrease in start up delay with increase in number of disks. Bandwidth utilization for varying load is ascertained in Fig. 3. The increase in bandwidth decreases time to download the entire video. Variations of load increase the time to download by 2% to 4%. Increase in bandwidth reduces the time to download by 75% reaching saturation at higher bandwidth rates.

6   Conclusions

An efficient storage mechanism has been proposed to balance the load in the video server. The system has a low delay of 2ms - 7ms for a varying load. The bandwidth is utilized efficiently and less time is required to download the videos. The feasibility of replication-on-demand is critically dependent on the replication bandwidth availability. We believe that the simplicity of their implementation and the flexibility they offer makes these policies especially attractive for implementation in scalable video servers. Our future work is to examine the possibility of providing fault tolerance along with balancing the load in a video server.

References

1. Choe, Y.R., Pai, V.S.: Achieving Reliable Parallel Performance in a VoD Storage Server Using Randomization and Replication. In: Intl. Parallel and Distributed Processing Symposium, pp. 1–10 (2007)
2. Golubchik, L., Muntz, R.R., Chou, C.-F., Berso, S.: Design of Fault-Tolerant Large-Scale VoD Servers: with Emphasis on High-Performance and Low-Cost. IEEE Trans on Parallel and Distributed Systems 12(4), 97–109 (2001)
3. Zhou, X., Xu, C.-Z.: Optimal Video Replication and Placement on a Cluster of Video-on-Demand Servers. In: Intl. Conference on Parallel Processing, pp. 547–555 (2002)
4. Huang, X.-M., Lin, C.-R., Chen, M.-S.: Design and Performance Study of Rate Staggering Storage for Scalable Video in a Disk-Array-Based Video Server. In: Intl. Workshop on Network and Operating Systems support for Digital Audio and Video, pp. 177–182 (2005)