A Scalable Video Server Using Intelligent Network Attached Storage[1]

Guang Tan, Hai Jin, and Liping Pang
Internet and Cluster Computing Center
Huazhong University of Science and Technology, Wuhan, 430074, China
{tanguang, hjin, lppang}@hust.edu.cn

Abstract. This paper proposes a new architecture, called intelligent network attached storage, for building a distributed video server. In this architecture, the data intensive and high overhead processing tasks such as data packaging and transmitting are handled locally at the storage nodes instead of at special delivery nodes. Thus an unnecessary data trip from the storage nodes to the delivery nodes is avoided, and a large amount of resource consumption is saved. Moreover, these ì intelligentî storage nodes work cooperatively to give a single system image to the clients. Based on the architecture, we design our admission control and stream scheduling strategies, and conduct some simulation experiments to optimize the system design. The simulation results exhibit a near linear scalability of system performance with our design. Some implementation issues are also discussed in this paper.

1. Introduction

The incredible progress of multimedia and network technology enables one to build *Video-on-Demand* (VoD) system delivering the digitized movies to PCs or set-top-boxes of its subscribers. In order to support hundreds or thousands subscribers, the server for VoD service must be a high performance computing system. While many systems are built on a single high performance computer, more and more people tend to run this service on a cluster system composed of off-the-shelf commodities for their cost-effectiveness and fault-tolerant capability, and this area has attracted increasing attention of the researchers.

In designing a video server, one major concern is to pump a huge amount of data from disks to many clients at the same time, which is a great challenge due to the relatively slow I/O facilities. Most of the existing schemes are to develop a highly efficient parallel file system (e.g., Tiger Shark File System [14], xFS [4]) to achieve a satisfactory I/O bandwidth. In these systems, data is partitioned into stripes and distributed across storage nodes. When needed, they go through one delivery node, or a proxy [19] in parallel, where they are encapsulated into packets according to a certain protocol like RTP (*Real-time Transport Protocol*), and finally sent to clients for playing back. A typical data flow process is shown in Figure 1.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-3-540-45812-8_28

[1] This paper is supported by Wuhan Key High-Tech Project under grant 20011001001.

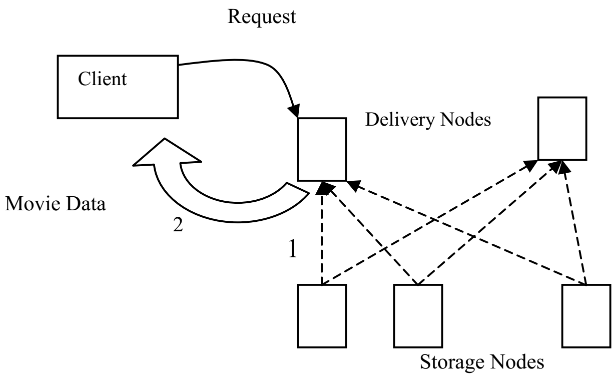


Fig. 1. Data flow in a traditional video server. In real implementations, the delivery nodes can also be storage nodes, or even machines independent of the server system.

One characteristic of this kind of systems is that the required data generally makes two trips on its way to clients: from the storage nodes to the intermediate nodes, and then to the clients. This makes the internal network or the processing capability of intermediate nodes potential bottlenecks for system performance. Even though high-speed network infrastructure like Gigabit Ethernet or Myrinet is adopted and the CPU speed is increasing rapidly, the scalability of this kind of systems is quite limited.

Our solution to this problem is to remove the first data trip between the storage nodes and the intermediate nodes (indicated by the dashed lines in Fig. 1) by offloading the latterís function to the former. If the storage nodes can work in a cooperative manner that makes the data from multiple independent nodes looks like from one machine, the clients can enjoy the video service without any loss on QoS. For the server side, a great amount of resource consumption is saved.

This idea is similar to the concept of active disk [1][23], intelligent disks [16], or network attached autonomous disks [2][3] in some ways. The common point is to push the basic processing closer to the data, providing a potential reduction in data movement through the I/O subsystem. This method is being applied to various applications such as database, data mining, image processing. Especially it is employed in multimedia streaming service [9]. Our structure however, has the following differences: while these systems have only small processing engines attached to the disks performing some simple operations, our storage nodes are capable of handling more complicated problems specific to one application, such as admission control, load balancing and automatic failover. Compared to some of these systemsí relying on clients to do some special task such as data block mapping and data pulling in predefined syntax, our system hides all the data locating details in itself, and therefore gives a single system image to the clients. This makes our scheme more adaptable in real internet world.

The rest of this paper is organized as follows: section 2 presents the system architecture and discusses some specific problems related to video server. Section 3 describes our simulation experiments and demonstrates some simulation results, reveling the relationship between server system parameters and the system performance.

Section 4 discusses several implementation issues. Section 5 evaluates some related work, and finally section 6 ends with a conclusion and the future work.

2 System Architecture

The system architecture is illustrated in Figure 2. The storage system in this figure can be realized by different hardware platforms, e.g., a regular workstation or active disks attached to a network [1]. In this paper we take the first approach and use the term disk and storage node with no difference.

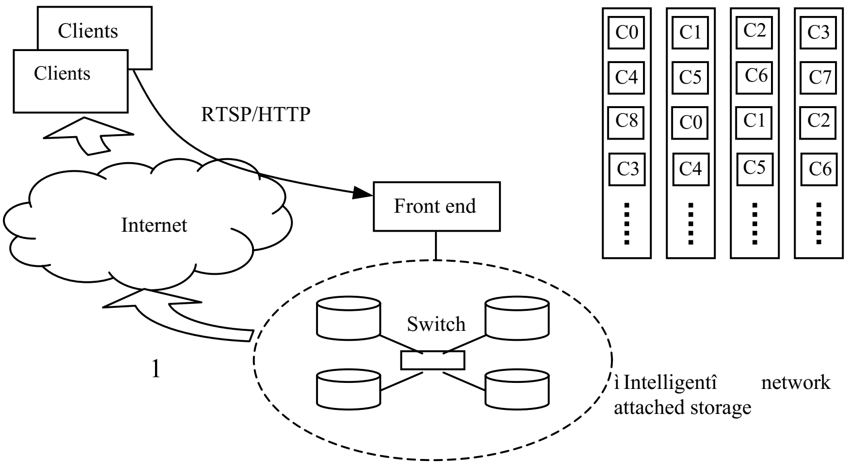


Fig. 2. System architecture. The video server system comprises of two major components: front-end machine and the ì intelligentî network attached storage.

The basic function of the front-end machine, which is the only entry-point of the system, is to accept the client requests and instruct storage nodes to retrieval the requested data and send it back to client directly. Usually it is implemented as a RTSP daemon or HTTP daemon, depending on the protocol adopted to carry the interaction commands between the clients and the server.

When the front-end machine accepts a clientís request, it first processes the authentication. If passed, the request will be broadcasted to the attached disks, which will translate it to a set of sub-requests and schedule them according to their own task scheduler. The front-end machine collects the scheduling results and makes a comprehensive decision, which is to be sent to the clients, and then it schedules the storage nodes to arrange their playing tasks.

Besides, the front-end machine takes care of redirecting the clientsí feedback to appropriate storage nodes to adjust the transmission quality. It is also the global manager of the UDP ports used by storage nodes. These problems will be further discussed in section 4.

2.1 Data Organization

The storage system is the container of movies and takes responsibility for packaging and transmitting of media data. Movies are partitioned into segments (movie clip) of equal playing time (except the last one) and interleaved across disks. Each segment has two replicas in order to achieve fault tolerance and load balancing. The segments of one movie are placed in a round robin fashion with the first one at a randomly chosen storage node, and the beginning nodes are different for one movieís two replicas. An example of the data layout is illustrated in Figure 2.

In this paper, each movie segment together with its processing code is regarded as an object. Similar to traditional concept of object, this abstract also has its own features. The attributes, data, and processing code of the object are all separately stored in disk as different forms: clip index file, clip file, and program file respectively. The clip index file is a text file recording all the clip items of one movie. It is named as xxx.index, where xxx is the corresponding movie file name, and the content of the index file is a table-like file illustrated in Table 1.

Table 1. An example of index file LifeIsBeautiful.mpg.index

Clip No.	Clip Name	Range	Buddy Location	Next Location1	Next Location2
0	LifeIsBeautiful.0.mpg	0 ñ 5:00	Node5	Node3	Node6
4	LifeIsBeautiful.4.mpg	20:00 ñ 25:00	Node5	Node3	Node6
Ö	Ö	Ö	Ö	Ö	Ö

In table 1, the Buddy Location refers to the location of another replica of the same movie segment, and the Next Location1 and Next Location2 point to the places of the next segments in playing time. These parameters serve as pointers when one object needs to interact with another one. The pointers link up all the objects into a well-organized data structure, as shown in Figure 3.

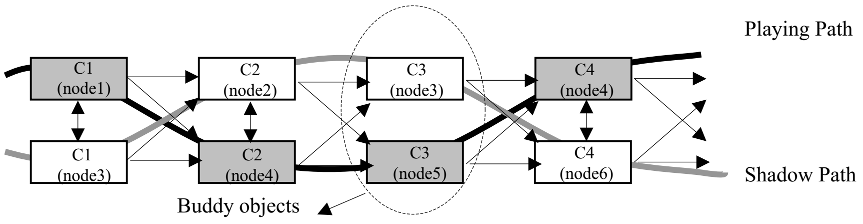


Fig. 3. Linked object lists. The sequence of grey blocks on the black curve represents the storage nodes to play the consecutive segments of one movie, and the blank blocks on the grey curve composes the shadow path.

With this structure, a movieís playing back is in fact a sequence of object operations: the first object retrieves and transmits its data, then it tells the next object to transmit next segment, and then next one, until all the objects finished its task. We call the sequence of objects participating in the real playing back a ìplaying pathî, and the remaining objects of this movie compose a ìshadow pathî. The shadow path accompanies the playing path and periodically pulling information like normal playing time, RTP sequence number, RTP timestamp, etc., as checkpoints. When one object crashes, its buddy object will resumes its work using the latest checkpoint data. We call the failover process ìpath switchingî. The playing path and shadow path is shown in Figure 3 and the path switching process is illustrated in Figure 4.

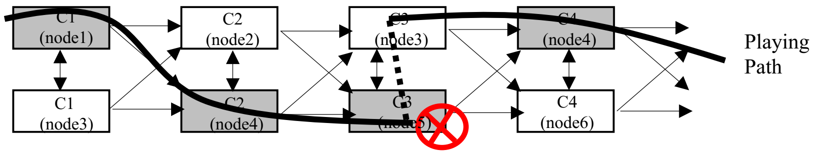


Fig. 4. Path switching upon a node failure. Node2 resumes the playing back of one movie when node5 fails in the middle of clip3.

2.2 Admission Control and Stream Scheduling

As an important issue for a system providing real-time service, admission control and resource allocation have been hot issues [13][15][17]. While it can enjoy some of the optimizations brought by these researches, our system has some particular features to take into account in its design.

Since a movie is partitioned into multiple segments, the task of playing a movie is divided into a set of sub-tasks, each representing the playing of one segment. A movie-playing request is translated into a set of sub-requests to the storage nodes according to the movie index files. All the sub-requests must be scheduled by the storage nodes before their corresponding sub-tasks are accepted, and the scheduling results jointly determine the result of the request.

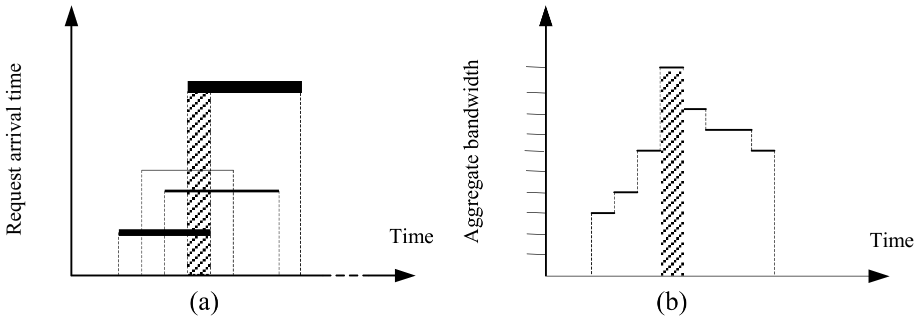


Fig. 5. Scheduling table and aggregate data bandwidth. In (a), each line represents a sub-task, or a streaming, and the width of the line indicates the resource requirement of this task.

To schedule sub-requests, each storage node maintains a scheduling table recording all the playing tasks being currently executed or to be executed. An example of the scheduling table is illustrated in Fig. 5(a). Our admission control and streaming scheduling are based on the peak value of the system load, defined as the aggregate data bandwidth of all the streams. Fig. 5(b) presents the aggregate data bandwidth vs. time. Denoting the bandwidth of individual task by b(T), the aggregate bandwidth at time *t* by B(t), and the beginning time and ending time of one task by bgn(T) and end(T), respectively. We define the peak value of system load in a time range of a task as:

$$L(T_{new}) = Max\ B(t) = \sum_{T \in \{T \mid T \in \Phi\ \wedge\ bgn(T) \le t \le end(T)\}} b(T) \quad + b(T_{new}),$$

$$bgn\ (T_{new}) \le t \le end\ (T_{new}) \qquad \textbf{(1)}$$

where Φ is the set of tasks that have been already arranged in the scheduling table.

If $L(T_{new})$ exceeds a certain upper bound (typically determined by the local disk I/O bandwidth and network interface bandwidth), this sub-request is rejected. Based on this, the admission control process is handled as follows: if one sub-request is rejected by both storage nodes owning the corresponding segments, the whole request is rejected. If only one storage node can admit this sub-request, the sub-request is assigned to this node for movie retrieval. If both storages nodes can admit this sub-request, the one with smaller $L(T_{new})$ will be selected as the operator of this segment in the movie playing back. All the scheduling results should be returned to the front-end machine for comprehensive decision. The algorithm is given in Figure 6.

```
RequestSchedule()
{
   int Decision = ACCEPTED;
  if( current overall data bandwidth + bandwidth of new request
     > maximum system outbound bandwidth)
     return REJECTED;
  Broadcast the new request to storage nodes;
  for (int i = 0; i < NumClips; i ++)
   Receive scheduling result and store it in result[i];
   for (int i = 0; i < NumClips; i ++)
       if result[i] = REJECTED {
            Decision = REJECTED;
            break;
       }
  if (Decision == ACCEPTED)
   Send PLAY commands to the storage nodes, asking them to ar-
   range the playing tasks;
  return Decision;
}
```

(a)

```
SubRequestSchedule()
{
  Receive request from front-end machine;
  Look up the clip table and map the request to a set of sub-
    request: subreq[nclip];
  for (int i = 0; i < nclip; i ++)
  /* A schedule result contains the evaluated load in the
   * time period of given sub-request, and a very large
   * value means the sub-request is rejected */
       result[i] = Schedule(subreq[i]);
  for (int i = 0; i < nclip; i ++){
       if local IP < Buddy IP of clip i
            Send result[i] to the buddy object;
  else {
       Receive result from the buddy object as result1;
       Compare result and result1, select a smaller one and re-
       turn it to the front-end;
  }
}
```

(b)

Fig. 6. Request scheduling and admission algorithm: (a) Algorithm for front-end machine, (b) Algorithm for storage node. These two parts work in a coordinated fashion to make a final decision regarding admission and stream scheduling. For simplicity, the error handling is omitted.

The above discussion is based on the assumption that the streams are constant bit rate and the local admission control adopts a threshold-based deterministic policy, that is, calculates the expected resource requirement using worst-case evaluation and gives a deterministic result regarding admitting or rejecting. Our decision model, however, is not limited to these assumptions. Since each storage node is an autonomous entity, it can use whatever algorithm to process the admission control, as long as it gives a deterministic or probability-based result to the front-end machine. If a probability-based policy is adopted, the final admission probability should be

$$P = \prod_{i=1}^{n} P_i \tag{2}$$

where *n* is the segment number of a given movie and P_i is the probability of admitting object *i* given by its storage node.

3 Design Optimization

To optimize the design, we develop a simulation model to study the relationship among some basic system parameters, such as segment length, number of storage nodes, and the overall system performance. We also conduct experiment to demonstrate the scalability of this architecture.

In the simulation, a process called front process represents the front-end machine accepting request from a request producer, and several other processes called back process represent the storage nodes scheduling and executing the ìplayingî tasks. There is no real media data processing and transmitting; and the load values are calculated from the scheduling table using the assumed parameters described below.

Though simulating in a simplified environment, we believe that the model can correctly reflect the behavior pattern of the major system components based on the assumption given below.

3.1 Simulation Assumptions, Parameters and Performance Metrics

Without loss of generality, we make some assumptions and parameterize the simulation model as follows:

(1) The arrival of clientsí requests is a steady Poisson stream with arrival rate = λ;
(2) Movie number $M = 200$, movie duration is conform to uniform distribution between 110 and 130 (minutes), with the average value $T = 120$; and the movie selection pattern is conform to Zipf distribution ($\alpha = 1$) [6];
(3) All requests ask for const-bit-rate streaming service, with data bandwidth = b;
(4) The admission control adopts a threshold-based policy. The threshold numbers of streams for the whole system and individual storage node are S_{server} and S_{disk}, respectively;
(5) Number of storage nodes is N;
(6) Request number $Rn = 1080$.

We use the following simulation metrics:

(1) Request satisfying rate R

It is defined as the ratio of accepted requests to all the client requests issued. Given a request arrival rate, this metric reflects the service capability of the server system. Another implication of this metric is the average system data throughput B, which can be derived from:

$$B = \lambda T \cdot R \cdot b \tag{3}$$

(2) Maximum concurrent stream number C

It reflects the peak performance of the system while guaranteeing the quality of service.

3.2 Simulation Results and Analysis

3.2.1 Scalability Test

We fix $S_{disk} = 45$, and the clip length = 8 (minutes). Varying the number of storage nodes and computing the systemís servicing capability, we have Figure 7.

The figures exhibit a near linear scalability of the servicing capability of our system. Using equitation (3), we also learn that the average data throughput has a linear scalability with the scale of the system.

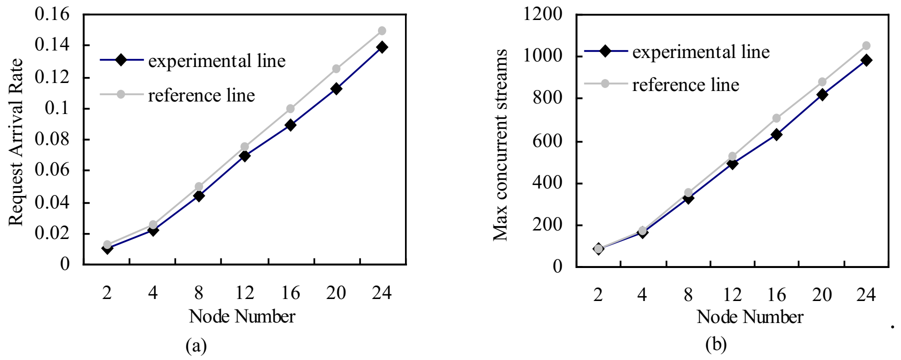


Fig. 7. System scalability vs. system performance. Here the Request Arrival Rate is the maximum arrival rate that makes the serviceís satisfying rate no less than 97.5%.

3.2.2 Relationship between System Parameters and System Performance

The purpose of this experiment is to find out the relationship between some basic system parameters and the overall system performance. We studied two parameters, clip length and replica number of movies, called replicate degree. For the former parameter we vary the clip length and computing the satisfying rate and number of maximum concurrent streams, showing in Figure 8.

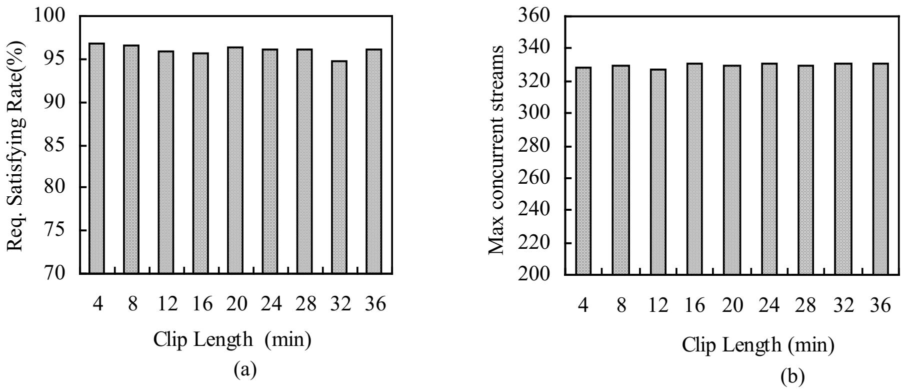
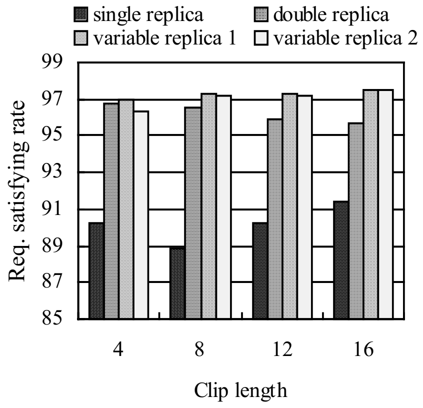


Fig. 8. Clip Length vs. system performance. Here $\lambda = 0.045$, $S_{disk} = 45$ and $N = 8$.

From Fig. 8 we can see the length of clip have only slight influence to the system performance. This phenomenon is probably due to our assumption that the client requests arrive as a steady Poisson stream, which has a characteristic that the numbers of request appearing in two different time ranges are independent stochastic variables. So a single long clip has the same effect as a group of short clips in assigning the playing task, though they occupy different periods of playing time in the time axis of one movie.

To find out the relationship between replicate degree and the system performance, we examine 4 different patterns for the replicate degree setting in simulation: (1) Single replica. Each movie has only one replica; (2) Double replica. Each movie has two replicas; (3) Variable replica 1. Each one of the top 10 (most frequently accessed) movies has replicate degree of 3, and the remaining movies have replicate degree of 2; (4) Variable replica 2. Each one of the top 10 movies has replicate degree of 3, each one of the top 11-70 has replicate degree of 2, and the remaining movies have replicate degree of 1.The simulation results are presented in Fig. 9.



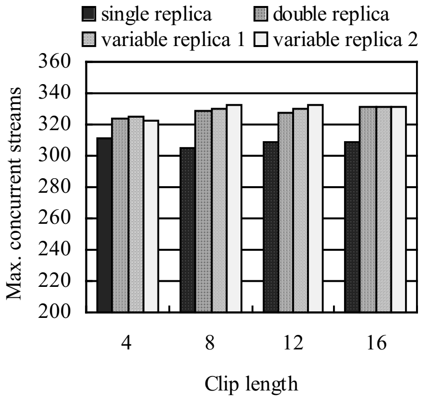


Fig. 9. Replicate degree vs. system performance

In Fig. 9 (a), there is a distinct improvement in system performance from single replica scheme to double replica scheme. This means that doubling the replica number will benefit much to system performance. From double replica scheme to variable replica 1 scheme, however, there is not much performance improvement, which suggests that the addition of replicas on the base of double replica scheme only bring quite limited benefit.

Compared with the second and third schemes, the variable replica 2 scheme has comparable performance. This implicates that the load balancing on the hot movies has significant influence on system performance, while the effort on balancing the load on cold movies has little gains. Therefore, the reduction in replica number of cold movies, which occupy the majority in the movie library, will have little negative influence on system performance. This result appears to be a valuable suggestion in real deployment: when the QoS for cold movies are not so critical, the cost on data storage can be reduced by (200-20-60)/400= 30% without performance degradation.

4 Implementation Issues

In a typical video service system, the server serves as a data provider and clients as data consumers. Most of the data transmission takes place from the server to clients, making it possible to scale up the one-way data bandwidth. However, some data (such as those required by RTCP) must be sent to the server as necessary information for adjusting the transmitting quality. These data should arrive at the appropriate storage node of the server via the front-end machine. As the data manipulation of one movie is changing from one storage node to another, the front-end machine needs to know where the operation on a movie is occurring when the feedback of a client comes. We add a port-mapping table in the OS kernel (our experimental platform is Linux Kernel 2.4.2). It receives messages from storage nodes and accordingly updates its mapping table so as to redirect the incoming UDP packets destined for some port to the appropriate receiver.

Another practical issue is the scalability of the system. Since the major responsibility of the front-end machine is to maintain the RTSP or HTTP connections, performing only very sparse interaction (like PLAY, PAUSE, TEARDOWN, etc. commands), and to direct the incoming RTCP packets. The overhead of the front-end machine is very low, and it can easily handle hundreds of concurrent VoD sessions. However, when the number of simultaneous clients exceeds 1000 or more, the front-end machine is very likely to become a performance bottleneck, as it is the single entry point of the whole system. A solution to this problem is to use multiple front-end machines to share the load. The available approaches include Dynamic DNS [8], One-IP technology [11], etc. Layer 4 switching technology can also be used to enhance the processing capability of front-end machine.

5 Related Work

Microsoftís Tiger [5][7] is a special-purpose file system for video servers distributing data over ATM networks. To play a video stream, Tiger establishes a multipoint-to-point ATM-switched virtual circuit between every node and the user. This mechanism achieves high parallelism for data retrieval and transmission. But the reliance on ATM network also brings a disadvantage: when receiving data from the server in a network environment other than ATM, front-end processors may be needed to combine the ATM packets into streams.

CMU-NASD [9][12] achieves direct transfer between client/drive in a networked environment. It exports a pure object-based interface to clients, and clients can contact with disks directly in a secure communications channel. Multimedia file accessing is experimented on this platform, but the emphasis is put on developing a file system, and VoD service as an Internet application conforming to a set of industrial standards is not considered.

C. Akinlar and S. Mukherjee [2][3] proposed a multimedia file system based on network attached autonomous disks. The autonomous disks, implemented using regular PC-based hardware, have specific modules called AD-DFS in their operating system kernels. When a client requests a file, the disks send data via their own network interfaces. But the client needs particular module in its OS kernel to understand the block-based object interface exported by AD-DFS. Like the NASD, the client must issue multiple network connections to the disks when accessing a file expanding several disks.

Parallel Video Server [20][21] employs an array of servers to push data concurrently to the client stations. Unlike our design to exploit parallelism among multiple streams, it attempts to achieve data retrieval and transmission parallelism within individual stream. One disadvantage of this architecture is the data distribution is not transparent to the clients. Client-side re-computing is needed to mask the data lose in playing back when server node fails [21].

6 Conclusion and Future Work

We introduced our new architecture for building a distributed video server, called intelligent network attached storage. In this architecture, the network attached disk acts as an ìintelligentî entity with its own policy to determine admission and stream scheduling. Each data segment is regarded as an object with its own processing method. The data of multiple objects is processed locally at the storage nodes and then delivered to remote clients in a consecutive manner among the storage nodes. By cooperation of movie objects the system achieves a single system image to clients. Based on this architecture, we studied several issues related to video service such as admission control, stream scheduling, data placement policy, and conduct simulation experiments, which shows a near linear scalability in servicing capacity.

Our future research is to further optimize the system performance. This includes studying the relationship between the movie accessing pattern and appropriate data striping and placement schemes, for example, system performance when the input requests are a non-stable Poisson stream, system performance in presence of object failures, etc. We also attempt to study the issues on software model. How should the software be partitioned between the delivery nodes and the storage nodes? What a protocol the storage nodes should use to interact with each other? How to support both real-time and non real-time applications effectively? To provide structural support for a wider range of data intensive applications based on this architecture, all these issues need to be explored.

References

1. A. Acharya, M. Uysal, and J. Saltz. ìActive Disksî. *Proceedings of International Conference on the Architectural Support for Programming Languages and Operating Systems.* 1998
2. C. Akinlar and S. Mukherjee. ìA Scalable Distributed Multimedia File System Using Network Attached Autonomous Disksî. *Proceedings of 8th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, 2000. Page(s): 180 -187

3. C. Akinlar and S. Mukherjee. ìBandwidth Guarantee in a Distributed Multimedia File System Using Network Attached Autonomous Disksî. *Proceedings of Sixth IEEE Real-Time Technology and Application 2000 Symposium*. RTAS 2000. Page(s): 237 ñ246
4. T. E. Anderson, M. D. Dahlin, J. M. Neefe, D. A. Patterson, et al. ìServerless Network File Systemsî. *ACM Transactions on Computer Systems*, February 1996
5. K. Argy. ìScalable multimedia serversî. *IEEE Concurrency*, Volume 6, Issue 4, Page(s): 8-10, Oct.-Dec. 1998
6. R. L. Axtell. ìZipf Distribution of U.S. Firm Sizesî. *Science*. Sept. 7, 2001, Vol. 293.
7. W. J. Bolosky et al. ìThe Tiger Video Fileserverî, *Proc. of Sixth Intíl workshop on Network and Operation System Support for Digital Audio and Video*, 1996.
8. T. Brisco, ìDNS Support for Load Balancingî, RFC 1794, http://www.landfield.com/rfcs/rfc1794.html
9. Carnegie Mellon University. ìExtreme NASDî. http://www.pdl.cs.cum.edu/extreme/
10. Cisco Local Director, Cisco Systems, Inc., http://www.cisco.com/univercd/cc/td/doc/pcat/ld.htm
11. P. Damani, P. E. Chung, Y. Huang, C. Kintala, and Y. Wang. ìONE-IP: Techniques for Hosting a Service on a Cluster of Machinesî. *Proceedings of 6th International World Wide Web Conference*, April 1997
12. G. A. Gibson, D. F. Nagle, K. Amiri, and et al. ìFile Server Scaling With network-attached secure disksî. *Proceedings of the ACM International conference on Measurement and Modeling of Computer Systems (Sigmetricsí97)*, June 1997
13. A. Hafid. ìA Scalable Video-on-Demand System Using Future Reservation of Resources and Multicast Communicationsî. *Computer Communications* 21 (1998), Page(s): 431-444
14. R. Haskin and F. Schmuck. ìThe Tiger Shark File Systemî. *Proceedings of COMPCON*, Spring 1996
15. X. Jiang and P. Mohapatra. ìEfficient Admission Control Algorithms for Multimedia Serversî. *Multimedia Systems*, 7:294-304, 1999
16. K. Keeton, D. Patterson, and J. Hellerstein. ìA Case for Intelligent Disks (IDISKs)î. *ACM SIGMOD Record*, September 1998
17. E. W. Knightly and N. B. Shroff. ìAdmission Control for Statistical QoS: Theory and Practiceî. *IEEE Network*, March 1999
18. J. B. Kwon, H. Y. Yeom. ìAn Admission Control Scheme for Continuous Media Servers using Cachingî. *Proceeding of IEEE International Performance, Computing, and Communications Conference, 2000*. IPCCCí00. Page(s): 456-462
19. J. Y. B. Lee. ìParallel video servers: a tutorialî. *IEEE Multimedia*, Volume 5 Issue 2, April-June 1998, Page(s): 20 ñ28
20. J. Y. B. Lee. ìConcurrent Push ñ A Scheduling Algorithm for Push-Based parallel Video Serversî. *IEEE Transactions on Circuits and Systems for Video Technology*. VOL. 9, No. 3, April 1999
21. J. Y. B. Lee. ìSupporting Server-Level Fault Tolerance in Concurrent-Push-Based Parallel Video Serversî. *IEEE Transactions on Circuits and Systems for Video Technology*. VOL.11, No.1, January 2001
22. Linux Virtual Server Project, http://www.LinuxVirtualServer.org/.
23. E. Riedel. *Active Disks ñ Remote Execution for Network-Attached Storage*. Doctoral Dissertation. School of Computer Science, Carnegie Mellon University. 1999