

# A Case for Grid based Video on Demand System

Subhabrata Bhattacharya    Ravi Chandra Nallan    Anirban Chakrabarti

*Software Engineering and Technology Laboratory  
Infosys Technologies Ltd.  
Bangalore 560 100, India*

*{Ravi\_Nallan, Subhabrata\_B, Anirban\_Chakrabarti}@infosys.com*

## Abstract

*The Video on Demand (VoD) services incorporate streaming of video over network and allow its subscribers to select videos and play them in near real time playback quality including interactive functions like Fast Forward, Rewind, Random seek, etc. VoD systems put a huge amount of overhead on the processing capabilities of the video server and need an equally huge amount of storage. These systems also demand a highly optimized network backbone for data transfer. Though lot of research have been carried out in the area of VoD distribution and network optimization, the problems mentioned above have received only cursory attention. Recently, research in the high performance community has led to the development of Grid Computing technologies for precisely the problems stated above. In this paper, we propose and develop the idea of integrating VoD servers with Grid computing and describe the system as Grid based Video on Demand (GDVoD) system. Prototype of GDVoD system has been developed and experiments have been carried out. The experiments highlight the fact that GDVoD system has low overhead in terms of computation without compromising the quality of the streamed video.*

## 1. Introduction

The advent of digitization has brought a major shift in paradigm in the context of compression, storage and communication of video data. This has resulted in diversifying usage of digital video across a wide spectrum of application domains. Aeronautics and Space research, Pharmaceuticals, Academics and Entertainment are a few to cite in these domains that are the key consumers of various forms of digital video technology. With Internet playing a pivotal role in communication, access to video information has become ubiquitous. Today's scenarios require, conferences to be held and its proceedings to be shared in the form of digitized video across various sites in near-real time, without significant loss of quality.

A trivial solution to the above would probably be to distribute the video file through direct download. However, such a setup has latency overhead as access to the information content can only start once the entire file is downloaded. This leads to the evolution of the concept of streaming [1],[2] which decomposes a media file into a stream of packets that could be transmitted efficiently over computer networks and on receipt at the client site, can be played without waiting for the complete set of media data. Different video streaming mechanisms take advantage of specialized network protocols like RTSP [3] etc.

Multimedia streaming finds its practical implementation in Video on Demand (VoD [4]) systems. These systems incorporate video streaming with complementary technologies enabling subscribers to select videos from a catalog and watch them in near-real-time playback quality unlike the traditional TV broadcast services. Further interactivity like Fast-forward, Rewind, Random seek, Pause [5] etc. could also be introduced at the subscriber location to facilitate virtual video playback. However, such an efficient VoD implementation requires significant computation power, adequate storage and abundant network bandwidth. Recently, research in the high performance community has led to the development of the Grid Computing [6],[7] technologies, where heterogeneous computing and networking infrastructure combine to form a single computing infrastructure. Grid computing provides resource on-demand and hence provide a high Return on Investment (ROI), as the resources can be shared as per need. This paper is an attempt to integrate Grid with the Video-on-Demand systems through the development of Grid Based VoD (GDVoD) system developed in SETLabs in Infosys Technologies, Bangalore (India).

## 2. Motivation and Related Works

In this section, we will illustrate the problems associated with many Video-on-Demand systems and motivate the use of Grid based Video-on-Demand system.

### 2.1 Motivation

The various techniques that perform compression on video files encode these files at certain rates (bit-rate). The higher the bit-rate, the better is the quality of the video and the higher the file size. Large files put more constraints on the network bandwidth during transfer. Therefore delivery of video through streaming is largely determined by the availability of network bandwidth.

Let us assume a video file has been encoded at a bit-rate of  $\mu$  kilobits/sec. This file can be played over a network without significant loss of quality, provided the network can furnish a consistent bandwidth  $\beta$ , for a period of time  $t$ , is

$$\beta t \geq \mu t + p$$

where  $p$  is the transmission payload.

Best-effort packet networks like the Internet cannot provide such consistent bandwidths and hence video must be re-encoded at a rate lesser than the available bandwidth and subsequently transmitted. In an effort to solve this problem, many VoD systems [8],[9] implement Real-time stream bit rate switching [10] or adaptive streaming [10] which adjust streaming according to the availability of network bandwidth. Such an order of adaptation to network bandwidth can be achieved either by (a) Video is stored as files pre-encoded in multiple bit-rates and transmitted subjected to the availability of bandwidth or (b) Video is encoded dynamically according to the changes of bandwidth

Strategy (a) would require a total storage space

$$S = \sum n_i \mu_i$$

for each video file, where  $n_i$  is the number of instances the file is encoded and  $\mu_i$  being the different pre-encoding bit-rates. For a Video on Demand system that needs to cater to  $N$  different video requests, the total storage space is at least  $NS$  storage units. Since these requests are random, a highly sophisticated and scalable storage system needs to be deployed.

Strategy (b) on the other hand involves encoding fragments of a video file at different bit-rates according to the fluctuations in transmission bandwidth and sending them in near-real time. As the number of requests surges, so does the network traffic

at the VoD server which drastically strangles the delivery performance unless a highly robust and scalable network backbone is used. Encoding video itself is a compute intensive process and for every switch to a new bit-rate, the system would need to have computing power in the order of several gigaflops/sec. Also efficient management of factors like interrupt load and physical memory is necessary.

Setting up hardware that resolves aforementioned issues incur huge installation and maintenance costs. This situation clearly motivates us to exploit the advantages of GRID computing to architect a VoD system. This also motivated us to develop the Grid based VoD (GDVoD) system in our labs. The prototype has been developed and illustrates the benefits of Grid computing. The GDVoD solution constitutes of a VoD server which is distributed across low cost commodity hardware. A Grid based architecture helps us to distribute

1. The computation power required during the phase of re-encoding a video file into streams of suitable encoding bit-rates
2. The storage infrastructure required for a media library
3. Usage of communication bandwidth

We also design a novel strategy to split a significantly large video file into multiple chunks that we call *shards* and distribute these shards across the Grid. Information about all these shards is maintained in the form of a catalog which is periodically updated whenever there is a change in distribution. The advantages for employing this kind of fragmentation strategy is threefold

1. Storage can be managed efficiently as it is distributed across the Grid infrastructure.
2. With videos divided across shards, streaming a particular sequence of a video which has more demand than the rest of the file becomes easier.
3. High availability can be ensured through an effective replication scheme.

### 2.2 Related Work

Extensive research has been done addressing the issues faced by Video on Demand systems. Hewlett Packard Laboratories have conceived an architecture for Modular Streaming Media Server Content Delivery Network (MSM-CDN) [11] which is primarily based on caching multimedia content at multiple distribution sites called “edges” that are being in close proximity to the points of delivery. Thus such an architecture

delegates streaming to suitable sites that perform the actual streaming to the clients. The edge servers (a) Alleviate the computation load on the central VoD server, (b) Decentralize storage of video files, and (c) Distribute network usage.

A slightly different notion called CoOpNet [12] from Microsoft Research which largely relies upon peer-to-peer based content distribution framework. A VoD setup that is implemented on CoOpNet, mandates clients to co-operate in streaming. Each client is capable of caching a content that it has downloaded from a server in the recent past and whenever the VoD server is overwhelmed with requests, it redirects them to suitable clients. To adapt to the dynamics of network, each stream is encoded in different bit-rates through Multiple Description Coding (MDC) [12] and on arrival of requests, these sub-streams are delivered to the client by a different peer.

Tai *et al.* propose another peer-to-peer architecture to support a VoD system called P<sup>2</sup>VoD [13]. Since peers perform streaming, the systems efficacy is broadly concentrated in reducing latencies (a) incurred while identifying and adding peers to the distribution framework, and (b) involved in failure recovery.

Architecture as discussed in Server-less VoD [14],[15] conceptualizes a VoD setup without a server where the clients store video files among them and stream these files to the requesting client. The architecture is highly scalable with respect to storage as more and more clients are introduced in the distribution network, the collective file storage capacity increases linearly.

The system ascribed in [12], has streaming servers that have close proximities with clients. Such an architecture demands (a) Installation and maintenance of powerful machines at the “edges”, (b) special mechanisms for synchronization of cached files at the “edges” whenever a video file is updated at the main server. True Peer-to-peer setups on the other hand, have different manageability concerns. A consistent quality in the delivery of streams is difficult to guarantee in these situations wherein different peers might have different capabilities to stream. It is to be noted that the GDVoD solution actually complements the existing solutions in this area.

The rest of the paper is organized as follows. In next section, we define the detailed architecture of our prototype with each component described in detail. Section 4 outlines our observations and discuss the performance test results. We conclude in the following section by pointing out the salient contributions and future work.

### 3. Architecture

In this section we discuss in detail about our GDVoD system. The GDVoD system (see Figure 3a) comprises of non-dedicated, inexpensive machines interconnected with each other through standard network backbone, which form the distributed VoD server responsible for delivering video to the clients. These machines form the nodes in the Grid infrastructure. A client is totally abstracted from the view of the actual Grid nodes that participate in the VoD delivery process. Since storage and processing tasks get distributed, the GDVoD system is able to achieve high computation and storage using inexpensive machines resulting in the lowering of the Total Cost of Ownership (TCO). The single point of contact between the client and the GDVoD system is a proxy. Any request for a VoD delivery would first arrive at the proxy. The proxy is responsible to relay the request to a scheduler, which in turn based on certain performance metrics (Section 3.4) and other request specific information (Section 3.4) selects a viable set of GEMs (Grid Enabled Mini-server) from a set of Grid nodes which deliver the content to the client through the proxy.

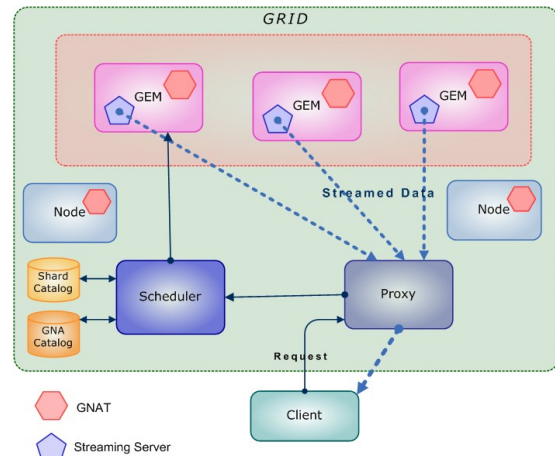


Figure 3a GDVoD High-level Architecture

The system comprises of the following major components.

*Grid Enabled Mini-server (GEM):* These are servers that store the video content and perform various other compute intensive tasks.

*Shard Creator:* It is responsible for splitting a particular video file into a number of fragments called shards.

*Streaming Server:* These are responsible for streaming video from content distributed across the grid.

*Catalogs:* These are repositories of information regarding resources pertaining to the grid and the video content stored in the grid.

*Grid Node Activity Tracker (GNAT):* These are minuscule agents that execute on each of the grid nodes to monitor system usage and perform various other activities.

*Scheduler:* This component is responsible for scheduling the video delivery process on the various GEMs.

*Proxy:* The proxy's main functionality is to validate and forward a client's request to the scheduler and deliver streamed video back to the client.

*Replicator:* The Replicator facilitates replication of shards across the grid to ensure redundancy for fail-over.

Each of these components are elaborated in the rest of this section.

### 3.1 Grid Enabled Mini-server

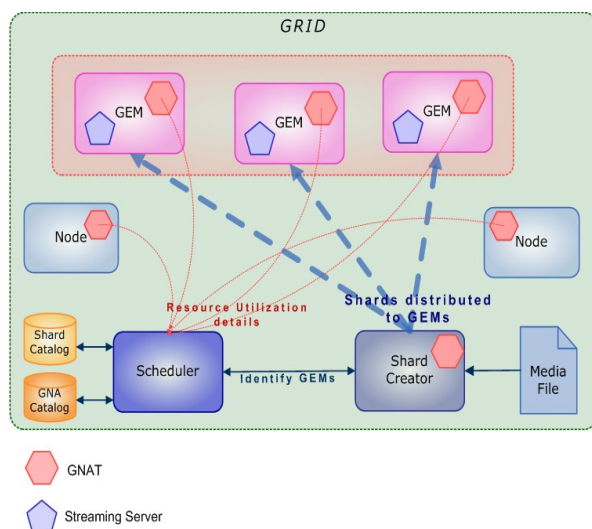
A Grid Enabled Mini-server or a GEM is one of the few selected machines that is primarily responsible for serving video content pertaining to a request. GEMs constituting a typical GDVoD system would need to be in close proximities and preferably be in a localized network to deliver high performance. A video file is fragmented into a number of shards with the help of a shard creator (section 3.2). These shards are stored across a set of GEMs identified by certain metrics called Grid Node Attributes (section 3.4). GEMs thus form source for streaming a video file.

### 3.2 Shard Creator

As discussed earlier in section 2.1, media files are encoded using certain algorithms in specific bit-rate to achieve good quality and compression. Hence any operation performed on the file that leads to change its size or structure, would require the knowledge of the encoding algorithm and the corresponding bit-rate. Since we intend to divide the media file into multiple shards first and then stream these individual *shards*, we would need to identify the bit-rate and algorithm that has been used to encode the file.

The shard creator works in conjunction with the various decoders [16] available for known video files to shred them into a sequence of entities which could not be decomposed further. These entities could be played independently as they retain the information that was used to encode the original video file. A contiguous sequence of these entities is called a *shard*. The entire process of shard creation needs high computation power and an equally huge storage capacity. A suitable grid node is identified consulting the GNA catalog (refer section 3.4).

The shard creator hence finds its place in the initial setup of the GDVoD system where a media file is split into shards and distributed over the Grid (refer Figure 3b). The shard creator works on the basis of the shard distribution algorithm which is described subsequently.



**Figure 3b Initial setup for GDVoD system**

*The shard creation and distribution problem:*

The shard distribution algorithm addresses the problem of calculating the size of a shard to be allocated to a grid node (GEM) in such a way that the cost of streaming that shard is minimal and the resource usage per GEM is directly proportional to the individual capacity of the GEM.

Let us assume that a particular media file has been encoded in  $\mu$  kilobits/sec and consists of  $N$  independent entities  $\{e_1, e_2, \dots, e_N\}$ . We also have  $M$  grid nodes  $\{G_1, G_2, \dots, G_M\}$ . Each of these grid nodes has capacities  $\{G_{C1}, G_{C2}, \dots, G_{CM}\}$ , where capacity  $G_{Ci}$  is overall capacity of a node for storing, processing and streaming the shard over the network.

Dealing with practical scenarios we have observed that  $N \gg M$  and to harness the potential of the grid, an efficient distribution scheme needs to be employed.

Let

$$\hat{E}_i = \{e_{i1}, e_{i2} \dots e_{iK_i}\},$$

where  $\hat{E}_i$  is a shard that has to be put on  $GEM_{i,}$  and  $K_i$  is the number of contiguous nodes entities in the shard  $\hat{E}_i$  (where  $\sum K_i = N$ )

our problem is to find  $\hat{E}_1, \hat{E}_2 \dots \hat{E}_m$  such that,

$$\bigvee_{i=1}^M \hat{E}_i / G_{ci} \text{ is minimal}$$

In other words, we have to minimize the usage of the resources at node  $G_i$

if  $S_{E_i}$  is the value (cost) of streaming the shard  $E_i$ , then we have to distribute the shards so that the cost of streaming the shard per node is minimal (relative to the capacity  $G_{ci}$  of the node  $G_i$ ).

(This can be proved to be a NP complete problem in nature and an approximation algorithm has to be applied.)

We propose a solution to the problem using the following algorithm,

Let the  $N$  contiguous independent entities which need to be accumulated as shard(s) be

$$e_1, e_2 \dots e_N \text{ with size } b_1, b_2 \dots b_N$$

and

$$B_1, B_2 \dots B_m \text{ be the Bits per Node,}$$

Where,  $B_i = G_{ci} ((\sum b_i + \Delta) / \sum G_{ci})$ , and  $\Delta$  adjusts the file size so that the shards get distributed evenly.

We find, the shards size per node is proportional to the ratio of the capacities  $G_{ci}$  to the capacity of that Node.

Algorithm *FindShards* ()

```

Start :
set i ← 1
set S ← 0
set  $b_{i,i}$  to ∞ //sentinel
while i < N
do
  Add  $b_i$  to S
  Find the grid node j with the best fit for S, i.e  $Min(B_j - S)$  where  $B_j > S$ 
  if found j and  $B_j - S$  is less than  $b_{i,i}$ 
  then
    set  $\hat{E}_i \leftarrow S$ 
    reduce  $B_j$  by  $\hat{E}_i$ 
  set S ← 0
end if
increment i by 1
done
if  $S > 0$  //some entity's left for distribution
then
   $\bigvee_{i=1}^M$  increment  $B_i$  by  $((S + \Delta) / \sum G_{ci})$ ,
  //distribute the error
  goto Start
end if

```

The algorithm works on the principle that higher the capacity of the node, larger the shard size it can afford. Hence the shard size is calculated by fitting the overall file size proportional to the ratio of the capacities of the Grid Nodes. The algorithm works by finding out the best fit for a given shard size on a node.

As the shard sizes are not equal, there is probability that a shard may be left without a node to get allocated. The worst case being the last shard is relatively small compared to the capacity of the node. To solve this, we iterate over the algorithm again, but by incrementing the file size by the left over shard size (and some adjusting  $\Delta$ ) and also increasing the capacity of each node by distributing the remaining shard. This increases the probability of getting a shard allocated to a node and also enables better fitting.

We show an example how this algorithm can be implemented,

Let the set of values  $b_1, b_2 \dots b_N$  be,

$b_1$	$b_2$	$b_3$	$b_4$	$b_5$	$b_6$	$b_7$	$b_8$	$b_9$	$b_{10}$
10	23	11	19	21	14	12	22	16	9

$b_{11}$	$b_{12}$	$b_{13}$	$b_{14}$	$b_{15}$	$b_{16}$	$b_{17}$	$b_{18}$	$b_{19}$	$b_{20}$
18	20	17	24	35	36	22	27	20	24

And  $\sum b_i = 400$

Where,  $B_i = \text{ceil} (G_{ci} ((\sum b_i + \Delta) / \sum G_{ci}))$

The Nodes with  $G_{ci}$  in the ratio,

20:30:40:60:50,  
and  $\sum G_{ci} = 200$

Nodes		20	30	40	50	60	Error
Iteration 1	Total	40	60	80	100	120	24
	Actual ( $B_i$ )	35	59	79	98	105	
	Sequence	$b_{15}$	$b_{7-10}$	$b_{11-14}$	$b_{1-6}$	$b_{16-20}$	
Iteration 2	Total	42	64	85	106	127	0
	Actual ( $B_i$ )	24	63	85	105	123	
	Sequence	$b_{20}$	$b_{1-4}$	$b_{5-9}$	$b_{16-19}$	$b_{10-15}$	

It is obvious that as the iterations increase the utilization of each node is done efficiently (note that the least effective node is used the least, hence using the most effective nodes better). By introducing a weight factor to the left over entity ( $S$ ) of the loop, we can increase the probability of finding the effective set  $\{\hat{E}_1, \hat{E}_2 \dots \hat{E}_m\}$  that distributes the load proportionally on the Grid.

### 3.3 Streaming Server

The streaming server is responsible for streaming the video content available to a Node. Hence the streaming server would reside on every other GEM to support streaming. We select a lightweight open source utility for this called ffserver [17]. The streaming server just needs to stream the shards instead of entire files to the proxy. Since network conditions tend to change unpredictably in grids streaming needs to adapt to the varying network conditions. The more adaptive is the streaming, the smoother is the receipt of the video at the client side. Usually advanced streaming servers have a feedback mechanism through which they can adjust the bit-rates of encoding of the streamed content. To incorporate this functionality in our prototype, we obtain a set of average network bandwidths that are being used to transmit data. Based on these bandwidth values we pre-encode each shard in constant bit-rates ranging from  $\mu_1 < \mu_2 < \mu < \mu_3 < \mu_4$  kilobits/sec where  $\mu$  being the original encoding bit-rate. Whenever the client's network connectivity declines, transmission is switched to a bit-rate lesser than the bit-rate transmission was being carried out. Similarly if the available bandwidth increases, we switch transmission to a higher bit-rate. This switch results in reduction of

jitters, thereby yielding to a smooth viewing though the client experiences some significant difference in quality. This technique requires a greater storage space as discussed in section 2.1 and is perfectly suited for a grid.

Dynamic or On-the-Fly Encoding [18] is another alternative to pre-encoded streams wherein media is encoded on the fly depending on the clients bandwidth requirements. Since encoding needs to be done in near-real time latency huge amount of computation power is needed and this demands a lot more than trite commodity hardware.

### 3.4 Catalogs

The requirements for a VoD system as discussed in section 2.1 help us to identify Grid Node Attributes (GNA). These attributes are

- Computation efficiency ( $E_c$ ), calculated in terms of the processor's architecture, class and clock speed
- Compute Load ( $L_c$ ), The current computation workload on the node
- Available Physical Memory ( $A_m$ ), estimated from the amount of physical memory available for user space processes
- Available Storage ( $A_s$ ) to be used for storing video files and
- Network Load ( $L_n$ ), measured in terms of active network connections and rate of data transfer that is currently taking place.

A GNA catalog keeps updated information about the GNA corresponding to all the grid nodes. A Grid Node Activity Tracker (discussed in section 3.5) installed in each of these nodes in grid facilitates this activity. This catalog is consulted whenever a decision related to performance need to be taken.

Another catalog known as the shard catalog contains pieces of information about a video file, its shards and their location. This catalog is consulted by the scheduler (discussed in section 3.6) to find the most suitable options that are available corresponding to a particular request for a video.

### 3.5 Grid Node Activity Tracker (GNAT)

Every node in the Grid has a tiny software agent called GNAT installed. A GNAT agent is instantiated with

the system start-up and use minimal system resources. Each GNAT has a predefined set of objectives that it performs at specified periodic intervals. However, depending on resource usage, it could be configured to temporarily suspend its activities to prevent starvation of other high priority processes and resume whenever there is sufficient resource availability. The GNAT mandates can be enlisted as

**Resource monitoring:** A GNAT continuously monitors the grid node on which it is executed and determines the GNAs corresponding to the grid node.

**Shard Management:** GNAs extracted from each Grid node aid to

(a) Select a grid node capable of decomposing a video file into shards and

(b) Distribute shards across a set of GEMs.

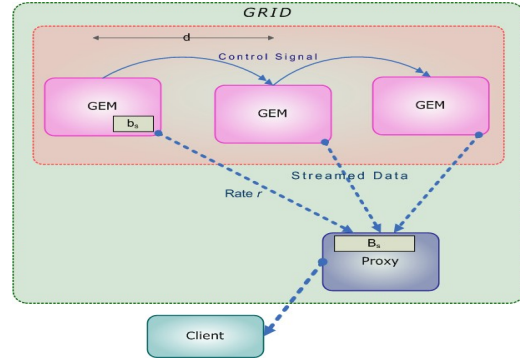
A GNAT resident on the target GEMs thus receives shards from the shard creator and places these to suitable locations in target GEMs.

**Catalog Maintenance:** The GNATs are also responsible for maintaining a catalog that contains information about the shards. Each GNAT communicates a tuple  $T$  {video\_identifier, shard\_sequence, GEM\_identifier} to a public Tuple-space [19]. The scheduler consults this Tuple-space based catalog to initiate the VoD delivery process.

**Replication initiation:** GNATs also initiate the process of replication (discussed in 3.8)

**GEM Synchronization:** Another necessary piece of activity done by the GNAT is to facilitate seamless streamed data transfer to the proxy. After a GEM finishes transfer of its data to the proxy, it signals the next GEM to initiate its share of transfer. This synchronization ensures a unified data flow to the client. The problem is illustrated with the following example.

*Optimized Synchronization problem:*



**Figure 3.5a Diagram illustrating synchronization between GEMs**

Let us assume that a set of GEMs  $\{G_1, G_2, G_3\}$  are serving a client request.  $G_1$  is about to finish its transmission and hence it needs to intimate  $G_2$  to start its transmission as soon as  $G_1$  has finished. Our goal is to find the time at which  $G_1$  would send the intimation signal to  $G_2$  so that the client would experience least transition gap.

Supposing  $B_s$  : size of the pre-roll buffer [20] which stores streamed video content before playback starts,  $b_s$  : total size of streamed data left to be sent to the buffer by  $G_1$ ,  $r$  : rate of data transmission from  $G_1$  to the pre-roll buffer and  $d$  : Time taken to send a packet between any two GEMs.

To state in words, the time taken to send  $b_s$  amount of data equals to sum of the time taken to send the intimation signal and time taken for the first frame to arrive at the pre-roll buffer

$$\begin{aligned} b_s/r &= d + 1/r \\ \Rightarrow b_s &= rd + 1 \end{aligned}$$

Introducing  $\delta$  as a degree of randomness in the rate of transmission, we get

$$b_s = rd + 1 + \delta$$

Hence data sent to pre-roll buffer before the intimation signal,

$$B_s \cdot (rd + 1 + \delta)$$

So time elapsed before a GEM sends an intimation signal to the following GEM to start transmission is given by

$$t_i = [B_s - (rd + 1 + \delta)] / r$$

### 3.6 Scheduler

Whenever a request for a video arrives at the scheduler, it firstly determines the locations of the shards for the requested video file. The shard catalog

helps the scheduler decide on the probable GEMs which would be required in the entire delivery process. In case there are multiple grid nodes available that contain the same shards, preference is given to the ones that outperform the others in terms of the GNA  $L_N$  (as discussed in section 3.4)

The scheduler being a normal grid node too has some storage space which could be configured for local caching. Certain video file shards that have the maximum demand over the others could be cached at the scheduler. There are a couple of advantages to have a facility like this (a) It eliminates the cost of streaming from multiple locations, (b) Reduces latency in delivering video content to the client through the proxy.

### 3.7 Proxy

The proxy plays an important role in delivering the streamed content to the subscriber. The client needs to request for a media to the proxy. The proxy's responsibility is to forward this request to the scheduler and transfer the streamed content delivered from the various GEMs. Thus it gives an impression to the client that media is delivered from one source. As the proxy is a single point of contact between the GDVoD system and the clients it could provide an efficient administrative tool to authenticate a client. Several policy related issues could be sorted out at this layer.

Once a request is validated, the proxy relays this to the scheduler which then starts processing it. The proxy is also equipped with a buffer to store the incoming streams from the GEMs. In case there is a temporary network outage between the client and the proxy, such a mechanism makes sure that content is not re-streamed.

Placing the proxy in the grid is an interesting issue in itself. While designing GDVoD we have come across three options in this regard.

(a) The simplest and the easiest of the three emphasizes on implementing a proxy per client. This nearly eliminates the transmission delay incurred while forwarding streamed content received at the proxy's buffer to the streaming client's pre-roll buffer. The client's buffer needs to be substantially large in order to avoid overflow.

(b) Another design suggests placing the proxy as an edge on the grid. This way many clients can access the same proxy and multiple proxies could be setup to

serve a group of clients. This architecture is highly reliable and scalable. Furthermore, the client doesn't need to have a significant pre-roll buffer size. However, this could increase latency in receiving pre-roll buffer data.

(c) The last design advocates placing the proxy as a part of the grid. The proxy behaves as a conventional server to the client request. A proxy with sufficient redundancy and fail-over mechanism can serve many clients.

We have observed that whenever there is a huge load on the GDVoD system, employing strategy (b) is a good option. The notion of implementing a proxy per client is recommended for a small number of clients.

### 3.8 Replicator

A portion of a video file or the entire file often experiences larger number of access requests than other portions of the same file or other files. Thus there is a need to guarantee availability of such a system. High availability of a particular video file or its shards could be ensured with sufficient redundancy systems. There are certain methods that are used to identify the media that is in demand. Most of these do so by tracking the number of requests for a particular media. Here we propose to introduce a replicator to facilitate redundancy or content mirroring across the various nodes in the grid. The granularity of replication is at portion level rather than file level. This provides more flexibility in replication.

As discussed in section 3.2, initially shards are distributed among the GEMs and the grid has only single instance of these shards. The GEMs to contain these shards are designated as primaries. The process of replication is initiated under the following circumstances:

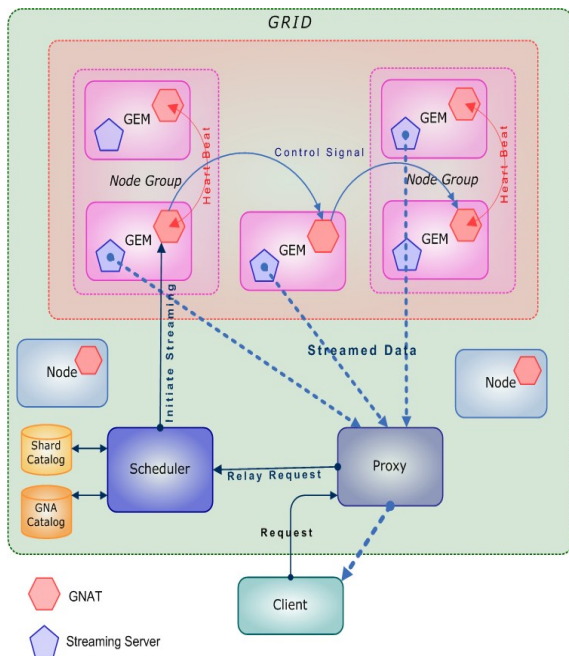
(a) Scheduler, being the only processor of all valid requests for videos, becomes aware of all the videos with the number of request exceeding a *configurable threshold*. Hence it intimates the respective GEMs associated with the video under consideration, to create replicas of the same video shards on various other GEMs.

(b) Whenever the load on the primary exceeds a *configurable threshold*, the GEM needs to back itself up before it breaks down, thereby initiating the replication.

In both these cases, it is the respective GNAT's responsibility to accomplish the replication process.

GNATs consult with the GNA catalog to find a suitable GEM for replication. Once a suitable GEM is located the primary's respective contents are transferred. The GNAT listening on the receiving GEM ensures that the contents are placed properly in the local file-system. Thus secondary GEMs are created which in conjunction with the primary GEM constitute a GEM group. Every node in this group communicates with each other GEM and hence follows a heartbeat protocol. This mechanism ensures the availability of the streamed content if a primary goes down. It also helps in distributing the load when the primary is overwhelmed by large number of requests.

Figure 3c gives a snapshot of GDVoD system at work. The diagram shows how replication guarantees high availability. The GEM's in a Node Group have the same shard; hence form the replicated nodes of the GEM. One can see that the streaming is done by a replicated GEM in the last node group.



**Figure 3c GDVoD Architecture in detail**

The flow can be described as,

Step I. Client requests the proxy for a media file

Step II. The request is relayed to the scheduler

Step III. The scheduler uses the Shard catalog to find the GEMs which can stream the requested media file

Step IV. The scheduler then initiates the streaming process by sending a request to the GEM with the first shard

Step V. Synchronization between the GEM's

(a) The GEM receives the request and starts streaming the shard it contains

(b) After calculating the delay, the GNAT on the GEM sends a control signal to next GEM, as notification for starting its stream.

(c) The Next GEM takes over and the loop continues till all the GEMs are finished with.

Step VI. The proxy receives the streamed content from the GEMs and forwards it to the client.

Step VII. The streaming client s/w on the client plays the stream as it comes.

The initial delay before the proxy forwards the stream to client is directly proportional to the delay between the GEMs stream transmission. Hence the buffer at the proxy is proportional to the delay between the successive transmissions. The paper proposes a method for calculating the delay effectively (section 3.5).

## 4. Observations

The GDVoD prototype is implemented keeping the dynamic capability of a grid system. The sections that follow discuss our observations.

### 4.1 Setup

The experiment is carried out in a miniature grid environment consisting of three independent machines serving as the GEMs (refer section 3.1). One of the GEM executes the scheduler and the proxy along with the streaming server. A separate machine located close to the proxy is configured to be the streaming client.

The performance is benchmarked against a single machine hosting the streaming server and a separate client. All tests are being carried out machines based on variants of Intel x86 processor hardware (Pentium 4 at 2.4GHz/Pentium III at 733MHz etc). These run different flavors of desktop Linux operating system configured with GNU/Linux 2.4.x Kernel. While conducting the tests, the machines were allowed to execute normal desktop applications simultaneously to

replicate a non-dedicated environment as observed in Grid systems.

Multimedia data, in the form of ac3/a52 [21] encoded, 128 kbps audio streams and rv10[22] encoded, VBR video streams, are wrapped in RealMedia container file format (.rm [23]). The video stream has a resolution of 640x480 at 25 frames per second. It is to be noted that not all multimedia file formats are capable of being streamed. There is no other specific reason to select ac3/a52 and rv10 encoding algorithms except for their

Ffmpeg [24] software is used currently because of its support for a wide range of codecs. Files of sizes starting from 50MB to 300M (in the order 50,100,150...)are created out of a regular multimedia file, precisely following the methods as described in the previous paragraph. Three shards of each file are created and relocated to a set of three GEMs. The streaming process is started and information pertaining to various parameters specific to each GEMs, like CPU time, Network bandwidth usage and latency involved during request processing are observed.

Mplayer [16] is used as a client for the streaming servers because of its wide acceptance in the multimedia community. As mentioned earlier in section 3.3, we rely upon ffmpeg to perform streaming.

For each run, every machine is allowed to perform in a steady state (ie. cases involving excessive load on the system in terms of memory,CPU,network traffic, are not taken under consideration). It is to be noted that all the streaming operations are done over HTTP (pseudo-streaming) since current implementations of the software do not have full support for true streaming protocols like RTSP and RTCP. Our prototype can exploit the services offered by these protocols once these are fully implemented.

## 4.2 Results

**CPU usage:** The cpu utilization is measured in terms of the number of seconds CPU spends while executing the streaming server process in user-mode and kernel mode. The observations in both cases (single and distributed streaming server) are tabulated as below in Table 4.2a and 4.2b respectively.

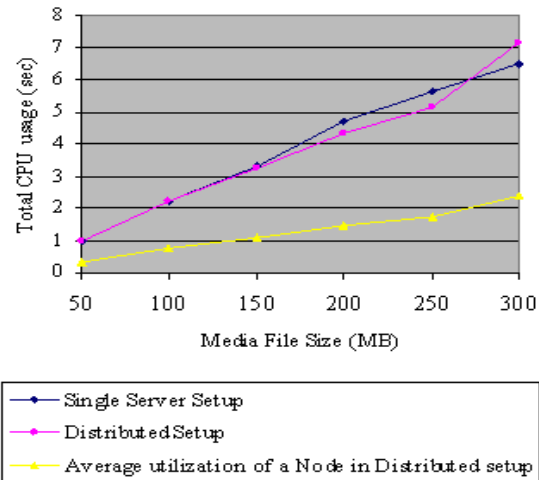
File Size (in MB)	CPU Usage (Real + User)
50	00.95
100	02.19
150	03.32
200	04.72
250	05.62
300	06.47

**Table 4.2a CPU time spent in non-distributed streaming server**

File Size (MB)	CPU Usage (Real + User) (sec)			Avg. usage (sec)	Efficiency
50	00.29	00.34	00.37	00.33	2.85
100	00.75	00.78	00.68	00.73	2.97
150	00.97	01.03	01.22	01.07	3.09
200	01.25	01.56	01.53	01.44	3.26
250	01.69	01.86	01.59	01.71	3.28
300	02.25	02.52	02.36	02.37	2.72

**Table 4.2b Total CPU time spent in distributed streaming server**

The average CPU usage in case of the distributed setup, is almost one third of the non-distributed counterpart. Efficiency of the former over the latter is computed by taking simple ratio of the per-CPU usages. This experiment reveals an important trend in terms of the efficiency of the entire system. The efficiency increases almost linearly with the file-size, however after reaching a threshold, it drops. It is inferred from further experimental insight that the efficiency could be increased if the streaming servers are increased too.



**Graph 4.2c CPU utilization in Single and Distributed setup**

**Network Bandwidth usage:** We measure the network bandwidth used by each system in terms of the number of network packets being transmitted by each of our components. Since packets are routed in a double-hop fashion (from the streaming server to the proxy, from the proxy to the client), TCP connection overhead becomes twofold. This overhead could however be reduced if lighter protocols are used.

By splitting each file into three shards and relocating them in three different GEMs lessens the number of packets transmitted by a single machine. Hence, network bandwidth consumed by a single machine diminishes by a factor close to 3.

**Overall request processing latency:** GDVoD introduces an overhead in terms of processing a particular request for a media file. There are certain instances when an overhead is introduced which could be explained as follows:

- a) Proxy forwarding client's request to scheduler
- b) Scheduler forwarding request to appropriate GEM hosting the streaming server
- c) Proxy forwarding streamed data to client.

There is a constant overhead (10-15%) which is the time required for connection setup between the scheduler and the first GEM, thread creation, and synchronization between the threads. This overhead will be amortized as the connections increase. It has been observed that as the file sizes increase, the average overhead reduces to 4%.

## 5. Conclusion

Video on Demand (VoD) systems have received tremendous attention over the last few years as entertainment industry is moving towards distribution of on-demand video content to the subscribers. VoD systems put an enormous amount of processing capabilities on the server especially if on-the-fly encoding is done. One way to tackle this would be to distribute pre-encode video files at different rates and store them resulting in a huge storage requirement. Therefore, there is a need to tackle the dual problem of computation capabilities and the storage requirements. Grid computing technologies is proving to be a good success story in to provide computation on-demand, and the concepts have been incorporated into the VoD systems. The system thus conceptualized and implemented is called GDVoD system. In this paper, the detailed description of such a system is provided which provides the scalability of the Grid systems without sacrificing the flexibility of a typical VoD implementation. More work needs to be done in the following areas:

(a) There is a need for an extensive management interface incorporating security and policy level issues in our future endeavor.

(b) More extensive research needs to be done to deliver live media using dynamic encoding techniques.

(c) Currently the GDVoD system is being extended to a full-fledged VoD solution. Initial experiments that are being carried out to evaluate the performance of the system seem encouraging.

## 6. References

- [1] Jane Hunter, Varuni Witana and Mark Antoniadis, "A Review of Video Streaming over the Internet", DSTC Technical Report TR97-10, August 1997
- [2] John G. Apostolopoulos, Wai tian Tan, and Susie J. Wee, "Video Streaming: Concepts, Algorithms and Systems", HP Laboratories Technical Report 260, 2002
- [3] H. Schulzrinne, A. Rao, R. Lanphier and M. Westerlund, "Real Time Streaming Protocol (RTSP)", RFC 2326, Internet Engineering Task Force, April 1998
- [4] Jari Peltoniemi, "Overview of Video on Demand systems", Tampere University of Technology Telecommunications Laboratory January 1995.
- [5] Miranda Ko and Irene Koo, "An Overview of Interactive Video On Demand System", The University of British Columbia, December 1996.
- [6] Ian Foster, "What is the Grid? A Three Point Checklist", Technical Report, Argonne National Laboratory & University of Chicago, July 2002.
- [7] Ian Foster and C. Kesselman, "The Grid: Blueprint for a New Computing Infrastructure", Morgan Kaufmann Publishers, July 1998
- [8] Ranga S. Ramanujan, Jim A. Newhouse, Maher N. Kaddoura, Atiq Ahamad, Eric R. Chartier and Kenneth J. Thurber, "Adaptive Streaming of MPEG Video over IP Networks", 22nd Annual IEEE International Conference on Local Computer Networks (LCN'97)
- [9] Charles Krasic, Jonathan Walpole and Wu-chi Feng, "Quality-Adaptive Media Streaming by Priority Drop", 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video (NOSSDAV 2003), June 2003
- [10] B. Vandalore, W. Feng, R. Jain and S. Fahmy, "A Survey of Application Layer Techniques for Adaptive Streaming of Multimedia", Journal of Real Time Imaging, volume 7, issue 3, pp. 221-235, June 2001.

- [11] Sumit Roy, John Ankcorn, and Susie Wee. "Architecture of a modular streaming media server for content delivery networks", IEEE International Conference on Multimedia and Expo, 2003.
- [12] V. N. Padmanabhan, H. J. Wang, P. A. Chou and K. Sripanidkulchai, "Distributing streaming media content using cooperative networking" in ACM/IEEE NOSSDAV, Miami, May 2002
- [13] Tai Do, Kien A. Hua and Mounir Tantaoui, "P2VoD: Providing Fault Tolerant Video-on-Demand Streaming in Peer-to-Peer Environment", IEEE International Conference on Communications (ICC 2004), Paris, June 2004
- [14] Jack Y. B. Lee and W. T. Leung, "Study of a Server-less Architecture for Video-on-Demand Applications", IEEE International Conference of Multimedia and Expo., August 2002
- [15] Jack Y. B. Lee and Raymond W. T. Leung, "Design and Analysis of a Fault-Tolerant Mechanism for a Server-Less Video-On-Demand System", in proc. of ICPADS 2002, pp. 489-494
- [16] Gabucino, Diego Biurrun and Jonas Jermann, "Mplayer (1) Linux man page" Internet URL, <http://tivo-mplayer.sourceforge.net/docs/mplayer-man.html>, 2006
- [17] Philip Gladstone, "FFserver documentation", Internet URL, <http://ffmpeg.sourceforge.net/ffserver-doc.html>, 2006
- [18] Ingo Busse, B. Deffner, and H. Schulzrinne, "Dynamic QoS control of multimedia applications based on RTP," Computer Communications, Jan. 1996
- [19] Greg Byrd, "Tuplespace Computing on the Grid", Dept. of Electrical and Computer Engineering, North Carolina State University, March 2001
- [20] Mark Kalman, "Analysis of Adaptive Media Playout for Stochastic Channel Models", Stanford University, March 2001.
- [21] Advanced Televisions System Committee, "ATSC Standard: Digital Audio Compression (AC-3), Revision A", Internet URL [www.atsc.org/standards/a\\_52a.pdf](http://www.atsc.org/standards/a_52a.pdf), 2006
- [22] Real Networks, "Real Video 10 Technical Overview", Internet URL [http://docs.real.com/docs/rn/rv10/RV-10\\_Tech\\_Overview.pdf](http://docs.real.com/docs/rn/rv10/RV-10_Tech_Overview.pdf), 2006
- [23] Fabrice Bellard, "FFMpeg (1) Linux Man page", Internet URL <http://linux.com.hk/penguin/man/1/ffmpeg-g.htm>, 2006
- [24] Real Networks, "RealMedia File Format (RMFF)", Internet URL, <http://www.multimedia.cx/rmff-htm>, 2000