An End to End Software Only Scalable Video Delivery System

Navin Chaddha[+], Gerard A. Wall* and Brian Schmidt[+]

[+]Computer Systems Laboratory, Stanford University, CA- 94305.

*Sun Microsystems Laboratory, Mountain View, CA- 94043.

Abstract

Precompressed video delivery systems commonly operate at fixed data rates. However, variations in the availability of network bandwidth and processor cycles are common in dynamic general purpose computing environments. Variability arises from the outright lack of resources (e.g. network bandwidth and cpu cycles), contention for available resources due to congestion, or a user's unwillingness to allocate needed resources to the task. Users of a scalable video delivery system have greater flexibility and therefore, the system can more effectively deliver video in the presence of system resource scarcity. This paper describes an end-to-end system combining a new scalable video compression algorithm, video delivery software, a software video decoder, and a market-based mechanism for the resolution of conflicts in providing video to the user.

1. Introduction

Sun Microsystems Laboratories and Stanford University are building a number of applications and services which require video storage, processing, and transmission as a component. Among the services are a video library and an interactive lecture distribution system. A hierarchical video storage system is being built at Sun Microsystems Laboratories which will use a combination of magnetic disk, CDROM, and digital tape to store a year of television news broadcasts and accompanying annotations. Using text, speech, and image search engines being developed at Sun Microsystems Laboratories, a user will be able to browse the material for relevant news segments which the user may then select for full review. The video storage server migrates the full resolution, full frame rate news stories based on their age and access history from disk to CDROM to tape, leaving lower resolution versions behind to support the browsing operation. If a news segment becomes more popular or important, the higher resolution can then be retrieved and stored at a more accessible portion of the storage hierarchy. We present the design of such a system here.

This paper presents an end-to-end scalable video delivery system for situations in which the encoder operates independently of the decoder's capabilities and requirements. It produces an embedded bit-stream from which different streams at different spatial and temporal resolutions can be easily extracted. Bandwidth scalability with a dynamic range of a few Kbps to several Mbps is provided. The embedded bit-stream produced is prioritized with bits arranged in order of visual importance. The algorithm also allows easy joint-source channel coding on heterogenous networks. The subjective quality of compressed images improves significantly by the use of perceptual distortion measures.

A typical application of scalable compression is multicast over heterogenous networks consisting of ATM, Internet, ISDN, and wireless networks having differing bandwidth capabilities, and hosting decoders with various spatial and temporal resolutions, etc. Scalable compression is also important in image browsing, multimedia applications, transcoding to different formats, and embedded television standards. In addition, it can be used to overcome congestion due to contention for network bandwidth, CPU cycles etc., in the dynamic environment of general purpose computing systems.

Most existing compression systems do not have the desired properties of scalable compression. Compression standards like MPEG-2 offer scalability to a limited extent and lack the significant range in bandwidth. There is a significant body of work relating to video servers, most of this research has focussed on scheduling policies for on-demand situations, admission control, and RAID issues. However, there has been very little work on end-to-end software only scalable video delivery systems.

Our work differs in that it provides an end to end software only solution. The overall system combines the scalable compression algorithm, a multiple-user video storage system, disk management, network transport, video delivery software, decoder and synchronization mechanisms. It also provides a market-based mechanism for the resolution of conflicts in providing an end-to-end scalable video delivery service to the user. The service is divided into three groups of components: preprocessing, media server, and media player.

This paper is organized as follows. Section 2 gives the problem statement. Section 3 describes the technical approach. Section 4 presents the scalable video delivery system. Section 5 describes the performance results, and we conclude in Section 6.

2. Problem Statement

The aim of this work is to provide scalable video delivery over a wide range of different networks (ATM, ISDN, ethernet). The target decoder system should be able to define the spatial resolutions (i.e. 160x120, 320x240, 640x480 pixels) and temporal resolution (1 to 30 frames per seconds). Bandwidth scalability, with a dynamic range of the video data from 10 kbps to 10 Mbps, is also a requirement. The video encoding should output an embedded stream from which different streams at different resolutions (both spatial and temporal) and different data rates can be extracted depending on the decoder capabilities and requirements.

The software-based video decoder should be able to operate using minimal CPU resources on a range of systems. Inside the network, it must be possible to easily scale the embedded video stream to fit into a lower bandwidth link or to adapt to congestion. In addition, there should be error resilience in the decoder algorithm to allow for communication errors, such as bit errors or cell loss. The end-to-end system also requires support for audio-video synchronization and a scalable, multiple-user video storage system. Finally, there should be a simple mechanism to transform the user's selection of a delivery bandwidth to choose the most appropriate point in the spatial resolution, temporal resolution, data-rate and quality space.

3. Technical Approach

To meet the goal of a low cost, scalable video delivery system, a new video compression algorithm has been developed. Other standard compression algorithms were rejected as inappropriate. For example, MPEG-2 lacks the dynamic range of bandwidth, is costly to implement in software and uses variable length codes which require additional error correction support. Our scalable compression algorithm produces an embedded bit stream that can easily be rescaled by dropping less important bits from the video stream. Then a low cost, software-based decoder of the scalable video stream has been developed which only performs table lookups and additions to decode a frame of video. We have implemented a disk server which uses careful layout and scheduling to support multiple users of prerecorded video streams. The disk server utilizes the embedded stream of video to scale to the appropriate network bandwidth. Finally, the system utilizes an existing media synchronization framework and an electronic market based mechanism to provide a complete solution for end-to-end video delivery.

The following subsections present the different parts of the system. Section 3.1 describes the encoding and decoding algorithms. Section 3.2 describes the rate scalability algorithm. Section 3.3 describes the decoder architecture. Section 3.4 presents the disk server. Section 3.5 describes the network layer and section 3.6 gives the costing structure.

3.1. Description of algorithm

The video coding algorithm is based on a Laplacian pyramid decomposition [1] (see Figure 1). The original 640x480 image is decimated (filtered and sub-sampled) to 320x240 and 160x120 pixels for encoding. The base 160x120 is compressed and then decompressed. The resulting decompressed image is upsampled and subtracted from the 320x240 pixel image to give an error image. The error image is compressed and transmitted. The 160x120 decompressed image is also upsampled to 640x480 pixels. Then it is subtracted from the original 640x480 image to give an error image which is compressed and transmitted. Thus the encoding stage consists of three image resolutions. The base layer transmitted has the compressed data for 160x120 pixels image. The enhancement layer has the error data for the 320x240 and 640x480 images.

The decoder can support up to three spatial resolutions i.e. 160x120, 320x240 and 640x480 (see Figure 2). It can further support any frame rate as the frames are coded independently. To decode a 160x120 image the decoder just decompresses the 160x120 image. To get 320x240 or 640x480 image the decoder first decompresses the base layer (i.e. 160x120) image and then upsamples it to the correct spatial resolution. The next step is to decompress the error data in the enhancement layer and add it to the upsampled base image.

3.2. Rate or Bandwidth Scalability

In order to achieve bandwidth scalability with an embedded bit stream we use vector quantization [2] as our quantization scheme. Embedded coding is essential in achieving many of the above goals. Vector quantization (across transform bands) is critical in achieving the remaining goals. Both embedded coding and vector quantization can be performed by tree-structured vector quantization (TSVQ). TSVQ is a successive approximation version of vector quantization (VQ) [2]. In ordinary VQ, the codewords lie in an unstructured codebook, and each input vector is mapped to the minimum distortion codeword. This induces a partition of the input space into Voronoi encoding regions. In TSVQ, on the other hand, the codewords are arranged in a tree structure, and each input vector is successively mapped (from the root node) to the minimum distortion child node. This induces a hierarchical partition, or refinement of the input space as the depth of the tree increases. Because of this successive refinement, an input vector mapping to a leaf node can be represented with high precision by the path map from the root to the leaf, or with lower precision by any prefix of the path. Thus TSVQ produces an embedded encoding of the data. If the depth of the tree is R and the vector dimension is k, then bit rates $0/k$, $1/k$, $\ldots\ldots$, R/k can all be achieved. To achieve further compression the index-planes can be run-length coded followed by entropy coding. Algorithms for designing TSVQs and its variants have been studied extensively. For a survey, see [2].

Instead of using the mean squared error as our distortion measure, we use subjectively meaningful distortion measures in the design and operation of our TSVQ. For this purpose we transform the vector using Discrete Cosine Transform (DCT), and then apply the following input-weighted squared error to the transform coefficients:

$$d_T(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \sum_{j=1}^{K} w_j (y_j - \hat{y}_j)^2 \tag{1}$$

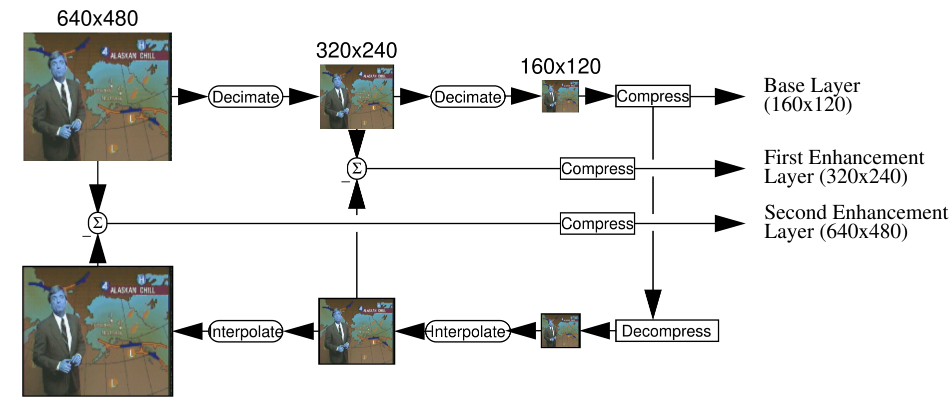

Figure 1. Block Diagram of the Laplacian Pyramid Encoding Algorithm

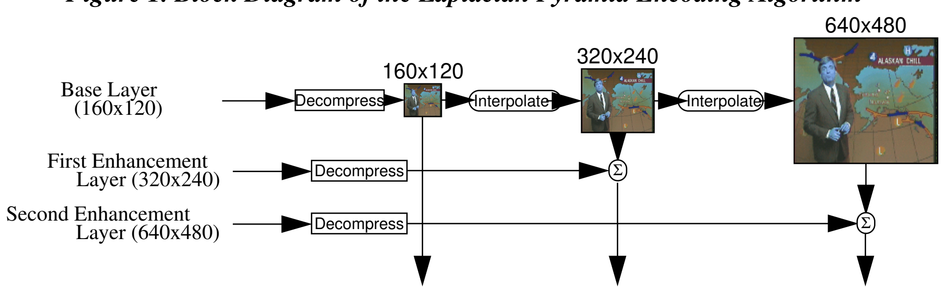

Figure 2. Block Diagram of the Laplacian Pyramid Decoding Algorithm

Here, y_j and $\hat{y}_j$ are the components of the transformed vector $\boldsymbol{y}$ and the corresponding reproduction vector $\hat{\boldsymbol{y}}$, and w_j is a component of the weight vector depending in general on $\boldsymbol{y}$. That is, the distortion is the weighted sum of squared differences between the coefficients of the original transformed vector and the corresponding reproduced vector.

The weights reflect human visual sensitivity to quantization errors in different transform coefficients, or bands. The weights are input-dependent to model masking effects. When used in the perceptual distortion measure for vector quantization, the weights control an effective stepsize, or bit allocation, for each band. When the transform coefficients are vector quantized with respect to a weighted squared error distortion measure, the weights $w_1, \ldots, w_K$ play a role corresponding to the stepsizes in the scalar quantization case. By incorporating the perceptual model into the VQ distortion measure, rather than into a stepsize or bit allocation algorithm, the weights can vary with the input vector while the decoder can still operate without the encoder transmitting any side information about the weights.

In the first stage of the compression encoder (Figure 1) an image is transformed using DCT. The second stage of the encoder forms a vector of the transformed block. Next the DCT coefficients are vector quantized using a TSVQ designed with a perceptually meaningful distortion measure. The encoder sends the indices as an embedded stream with different index planes. The first index plane contains the index for the rate $1/k$ TSVQ codebook. The second index plane contains the additional index which along with the first index plane gives the index for the rate $2/k$ TSVQ codebook. The remaining index planes similarly have part of the indices for $3/k, 4/k, \ldots\ldots, R/k$ TSVQ codebooks respectively. The advantage of this encoding of the indices is that it produces an embedded prioritized bit-stream. Thus rate or bandwidth scalability is easily achieved by dropping index planes from the embedded bit-stream. The decoder can use the remaining embedded stream to index a TSVQ codebook of the corresponding rate.

Frame-rate scalability can be easily achieved by dropping frames as there is no inter-frame compression in the algorithm right now. The algorithm further provides a perceptually prioritized bit-stream because of the embedding property of TSVQ. There is currently no motion estimation or conditional replenishment in the system. For future work these schemes will be incorporated in the system.

3.3. Decoder Architecture

The decoder of our video system is very simple. The decoder uses the indices from the embedded bit-stream to lookup from a codebook which is designed to use the processor cache efficiently. The process used for decoding the video stream consists of loading the codebooks into the processor cache and performing lookups from it. The base layer is obtained by performing lookups while the enhancement layers are obtained by performing lookups of the base and error images followed by addition. All operations of the decoder are performed beforehand i.e. by preprocessing.

The TSVQ decoder codebook has the inverse DCT performed on the codewords of encoder codebook. Thus at the decoder there is no need for performing inverse block transforms. Color conversion i.e. YUV to RGB conversion is also performed as a pre-processing step by storing the corresponding color converted codebook. To display video on a limited color palette display we color quantize the resulting codewords of the decoder codebook using the color quantization algorithm proposed by Chaddha et al [3]. This is achieved by forming a RGB or YUV color vector from the codewords of the codebook and color quantizing them to the required alphabet size. Thus the same embedded index stream can be used for displaying images on different alphabet decoders which have the appropriate codebooks with the correct alphabet size. (1-bit to 24-bit color)

3.4. Disk Layout

The disk layout consists of laying the video as two streams: base layer and enhancement layer streams. For our system we have not stored the error signal for the 640x480 resolution, since fairly good quality video was provided by bilinear interpolation of the 320x240 resolution images.

The base layer data is stored as a separate stream on the disk subsystem from the enhancement layer data. This allows the system to admit more users if fewer users choose to receive the enhancement layer data. The base layer data is stored with the following hierarchy:

1. Frames: Data for each frame is stored together. Each frame has a set of index planes corresponding to different number of bits used for the lookup.

2. Scalable Stream: The compressed stream consists of lookup indices with different number of bits depending on the bandwidth and quality requirement. The lookup indices for each frame are stored as groups of index planes pre-formatted with application level headers for network transmission. The 4 most significant bits of the lookup indices are stored together as the first section of the frame block. Then 4 additional 1-bit planes of lookup are stored in sequence as separate sections of the frame block to provide lookup indices with 4, 5, 6, 7, 8 bits respectively. The different lookup indices provide data streams with different bandwidth requirements. The server fetches the base signal frame block from the disk, transmits the selected sections on the network leaving the repacking of the bit planes into lookup indices to the receiving application.

The error data is placed similarly as another data stream. The lookup indices are stored as the most significant 2 bits of the lookup indices in the first section for each frame block. Then again the second 2 bits of the lookup indices as the second section. Then the 4 additional 1-bit sections of lookup indices are stored to provide lookup indices with 2, 4, 5, 6, 7, 8 bits respectively.

The video server uses RAID-like techniques [4] to stripe each (data stream) across several drives. The design allows for recovery from failure of any single disk without diminishing the capacity of the server. Because of the RAID approach, there is no restriction on the number of active users of a given title, as long as they can be accommodated within the servers's total bandwidth. That is, the usage can range from all active users receiving the same title at different offsets to all receiving different streams.

The streams of base and enhancement layer data are striped in fixed size units across the set of drives in the RAID group with parity placed on an additional drive. The selection of the parity drive is fixed since data updates are extremely rare compared to the number of times the streams are read. The current striping policy keeps all of the lookup indices for an individual frame together on one disk; while this costs some loss of storage capacity due to fragmentation, this policy allows for ease of positioning when a user is single stepping or fast forwarding their display. Use of parity on the stripe level allows for quick recovery after a drive failure at the cost of using substantially more buffer space to hold the full xor recovery data set.

3.5. Network Layer

The video server utilizes the planar bit stream format directly as the basis of the packet stream in the network layer. The embedded stream bits plus the application packet header are read from the disk and transmitted on the network in exactly the same format. For example, the base video layer has the four most significant bits of the lookup indices stored together so those bits are transmitted as one 2440 byte packet and each additional index bit plane of the less significant bits is transmitted as a separate 640 byte packet. The header contains a frame sequence number, nominal frame rate, size, a virtual time stamp, and a bit plane type specifier sufficient to make each packet an identifiable stand-alone unit. The server uses the self identifying header to extract the each bit plane group packet from the striped frame data retrieved from the disk subsystem.

The server also uses the sequence and rate information in the header as the means to pace the network transmission and disk read requests. The server uses a feedback loop to measure the processing and delivery time costs of the disk reads and queuing the network packets for transmission. The server then uses these measures to schedule the next disk read and packet transmission activities to match the video stream's frame rate (i.e. at X milliseconds in the future start transmitting the next frame of video). The server can moderate the transmission rate based on slow down/speed up feedback from the decoder.

The video decoder is responsible for the reassembly of the lookup indices from the packets received from the network. In the event of the loss of one of the less significant index bit plane packets, the decoder uses the more significant bits to construct a shorter lookup table index yielding a lower quality but still recognizable image.

The use of separately identified packets containing index bit planes makes it possible for networks to easily scale the video as a side effect of dropping less important packets. In networks providing QOS qualifiers such as ATM, multiple circuits can be used to indicate the order in which packets should be dropped (i.e. the least significant bit plane packets first). In an IP router environment, packet filters can be constructed to appropriately discard less important packets. For prioritized networks the base layer will be sent on the high priority channel while the enhancement layer will be sent on the low priority channel. In order to provide error resiliency the use of a fixed-rate coding scheme with some added redundancy, allows robustness in the face of packet loss.

The server supports two usage scenarios:

1. **Point-to-Point demand**: In this case each destination system decoder presents its specific requirements to the server. The server then sends the selected elements of the embedded stream across the network to the destination. A separate network stream per destination allows the user to have VCR style functionality such as play/stop/rewind fast forward/fast reverse. If congestion occurs on the network, then the routers and switches can drop packets from the embedded stream to give a lesser number of lookup bits.

2. **Multicast**: In this case the server puts out the entire embedded stream for the different resolutions and rates onto the network as a set of trees. The server has no idea about the decoders at the destinations. There may be one to eleven trees depending on the granularity of traffic control desired. The primary traffic management is performed during the construction of the unicast trees, by not adding branches of the trees carrying the less important bit streams to the lower bandwidth networks. The network in this case takes care of bandwidth mismatches by not forwarding packets to the networks which are not subscribed to a particular tree. Switches and routers can still react to temporary congestion by dropping packets from the embedded stream to deliver fewer bits of lookup.

3.6. Audio Subsystem

The delivery system treats the audio track as a separate stream which is stored on disk and transmitted across the network as a separate entity. The audio format supports multiple data formats from 8 KHz telephony quality (8 bit mu-law) to 48 KHz stereo quality (2 channel, 16 bit linear samples) audio. Most of the video clips on the current system have 8 KHz telephony audio since the intent is to be able to distribute the material over medium to low bandwidth networks. The server has the capability to store separate high and low quality audio tracks and to transmit the audio track selected by the user. Since the audio transits the network on a separate circuit, the audio can easily be given a higher QOS than the video streams. Rather than load the networks more with duplicate audio packets such as [8], we ramp the audio down to silence when packets are lost or overly delayed.

3.7. Audio/Video Synchronization

Since audio and video are delivered via independent mechanisms to the decoding system, the two streams must be synchronized for final presentation to the user. At the decoder, the receiving threads communicate through the use of a shared memory region, into which the sequence information of the current audio and video display units are written. Since the human perceptual system is more sensitive to audio dropouts and audio is difficult to temporally reprocess, the decoder uses the audio codec as the master clock for synchronization purposes. As the streams progress, the decoder threads post the current data items' sequence information onto the "blackboard", and the slave threads (such as the video decoder) use the posted sequence information of the audio stream to determine when their data element should be displayed. The slave threads then delay until the appropriate time if the slave is early (more than 80 milliseconds ahead of the audio). If the slave data is too late (more than 20 milliseconds behind the audio), then it is discarded on the assumption that continuing to process late data will delay more timely data. The video decoder can optionally measure the deviation from the desired data delay rate and send speed up and slow down indications back to the video server. This process synchronizes streams whose elements arrive in a timely fashion and does not allow a slow stream to impede the progress of the other streams.

3.8. Costing Structure

In the event of scarcity of resources, some global prioritization of user requests must take place, or overload collapse is likely. This system utilizes payment for services and resources as the means of defining the overall value of each resource allocation decision. Given these values, a total ordering of the user requests can be made and the less important requests can be dropped. The user specifies what he or she is willing to pay for a given service; this proposed payment along with the required resources (network and disk bandwidth) are submitted to an electronic market which uses micro-economic models to decide what amount of bandwidth resource is available to the user [5]. For that particular bandwidth a table is indexed to find the best possible combination of spatial resolution, frame rate and data rate (number of bits of lookup to be used) to give the best quality of decompressed video. This table is built using a subjective distortion measure [6]. The user also has the option of specifying the spatial resolution, frame rate and bandwidth directly.

4. Scalable Video System

The overall system combines the compression algorithm, disk management, network transport, decoder and synchronization mechanisms to provide an end to end scalable video delivery service. The service is divided into three groups of components: preprocessing, media server, and media player.

The preprocessing components are audio capture, video capture, video compression, and a data stripping tool. The video is captured and digitized using single step VCR devices. Then each frame is compressed off-line (non-real time) using the encoding algorithm. Currently, it takes about one second on a Sparc 20 Workstation to compress a frame of video data and the single step VCR devices can step at a one frame per second rate so capture and compression can be overlapped. The audio data is captured as a single pass over the tape. The audio and video time stamps and sequence numbers are aligned by the data striping tool as the video is stored to facilitate later media synchronization. The audio and video data are striped onto the disks with a user-selected stripe size. Currently, all of the video data on the server uses a 48 kilobyte stripe size since 48 kilobytes per disk transfer provides good utilization at peak load with approximately 50% of the disk bandwidth delivering data to the media server components.

The media server components include a session control agent, the audio transmission agent, and the video transmission agent. The user connects to the session control agent on the server system and arranges to pay for the video service and network bandwidth. The user can specify the cost he/she is willing to pay and an appropriately scaled stream will be provided by the server. The session control agent then sets up the network delivery connections and starts the video and audio transmission agents. The session control agent is the single point of entry for control operations from the consumers remote control, the network management system, and the electronic market. The audio and video transmission agents read the media data from the striped disks and pace the transmission of the data onto the network. The video transmission agent scales the embedded bit-stream in real-time by transmitting only the bit planes needed to reconstruct the selected resolution at the decoder. For example, a 320x240 stream with 8 bits of base, 4 bits of enhancement signal at 15 frames per second will transmit every other frame of video data with all 5 packets for each frame of the base and only two packets containing the four most significant bits of the enhancement layer resulting in 864 kilobits of network utilization. The server sends the video and audio either for a point-to-point situation or a multicast situation.

The media player components are the software based video decoder, the audio receiver, and a user interface agent. The decoder receives the data from the network and decodes it using lookup tables and places the results onto the frame buffer. The decoder can run on any modern microprocessor without the CPU loading significantly. The audio receiver loops reading data from the network and queuing up the data for output to the speaker. In the event of audio packet loss, the audio receiver will ramp the audio level down to silence level and then back up to the nominal audio level of the next successfully received audio packet. The system performs media synchronization [7] to align the audio and video streams at the destination. End to end feedback is used in the on demand case to control the flow. In the multicast case, the destinations are slaved to the flow from the server with no feedback. The user interface agent serves as the control connection to the session agent on the media server passing flow control feedback as well as the user's start/stop controls. The user can specify the cost he or she is willing to pay and an appropriate stream will be provided by the system.Figure 3 shows the block diagram of the system. The entire scalable system makes collaborative video over hetrogeneous networks possible without any special purpose hardware support.

5. Performance Results

A prototype of the system described in this paper has been implemented at Sun Microsystems Labs. The video data rate varies from 19.2 kbps to 2 Mbps depending on the spatial and temporal requirement of the decoder and the network bandwidth available. The PSNR varies between 31.63 dB to 37.5 dB. Table 1 gives the results for the decoding of a 160x120 resolution video on a Sparc Station 20. It can be seen from Table 1 that the time required to get the highest quality stream (8-bit index) at 160x120 resolution is 2.45 ms per frame (sum of lookup and packing time). This corresponds to a potential frame rate of 400 frames/sec. Similarly Table 2 gives the results for the decoding of a 320x240 resolution video on a Sparc Station 20. It can be seen from Table 2 that the time to required to get the highest quality stream (8-bit base index and 8-bit first enhancement layer index) at 320x240 resolution is 7.76 ms per frame (sum of lookup and packing time). This corresponds to a potential frame rate of 130 frames/sec. Table 3 gives the results for the decoding of a 640x480 resolution video again on a Sparc Station 20. It can be seen from Table 3 that the time to required to get the highest quality stream (8-bit base and 8-bit enhancement layer) at 640x480 resolution is 24.62 ms per frame (sum of lookup and packing time). This corresponds to a potential frame rate of 40 frames/sec.

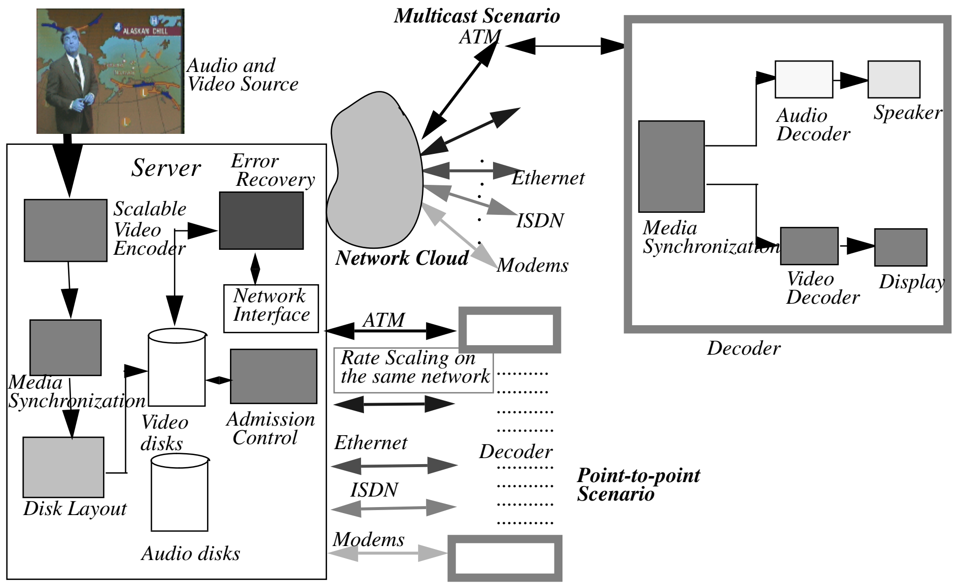

Figure 3. Block Diagram of the Scalable Video Delivery System

Table 4 shows the results for each individual disk for 160x120 resolution video. It can be seen from the Table that to get the highest quality stream (8-bit base) at 160x120 takes 5.60 ms of CPU time and an average CPU load of 2% on a Sparc 20 workstation. The average disk access time per frame is 16ms. Similarly Table 5 shows the results for each individual disk for 320x240 resolution video. It can be seen from the Table that to get the highest quality stream (8-bit base and 8-bit enhancement layer) at 320x240 takes 12.73 ms of CPU time and an average CPU load of 7% on a Sparc 20 workstation. The average disk access time per frame is 18ms.

6. Conclusions

In this paper we have presented a low cost, end-to-end scalable video delivery system combining a new scalable video compression algorithm, video delivery software, multiple-user video storage system, a software video decoder, and a market-based mechanism for the resolution of conflicts for shared resources which occur in providing streams of video to a community of users. In contrast with existing schemes, this approach provides clients with the ability to trade off video quality for system resources, permitting a much higher degree of overall value to be delivered with a given configuration of hardware.

The video encoding process creates an embedded video stream from which different streams at different resolutions (both spatial and temporal), and different rates, can be extracted depending on the capabilities and requirements of the decoders. In this system, the decoding subsystem defines the spatial and temporal resolutions of its displayed video stream (i.e., either 160x120, 320x240, or 640x480 pixels per frame, and from 1 to 30 frames per second). The various video quality specifications result in communications bandwidth scalability with a dynamic range from 10 Kbps to 10 Mbps. A low cost, software-based decoder of the scalable video stream has been developed which primarily uses table lookups and additions to decode frames of video. A disk-based video server has also been implemented which makes use of careful layout and scheduling to support multiple clients of the prerecorded video streams. In addition, the system provides support for media synchronization and makes use of an electronic-market-based mechanism to provide a complete solution for scalable end-to-end video delivery.

7. Acknowledgments

This work was conducted as a summer project of Navin Chaddha at Sun Microsystem Laboratories. Special thanks to Duane Northcutt and James Hanko of Sun Microsystem Laboratories and Prof. Anoop Gupta and Prof. Teresa Meng at Stanford University for many fruitful discussions.

8. References

1. N. Chaddha, “An efficient algorithm for scalable video compression with software only decode,” Technical Report Sun Microsystems, September 1994.

2. A. Gersho and R.M. Gray, Vector Quantization and Signal Compression, Kluwer Academic Press, 1992.

3. N. Chaddha, et al. “Fast Vector Quantization Algorithms for Color Palette Design Based on Human Vision Perception,” accepted for publication IEEE Transactions on Image Processing.

4. F. Tobagi, et al., “Streaming RAID- A disk array management system for video files,” Proc. ACM Multimedia 1993.

5. M. Miller, “Extending markets inward,” Bionomics Conference, San Francisco, Oct. 1994.

6. N. Chaddha and T.H.Y. Meng, “Psycho-visual based distortion measures for image and video compression”, Proc. of Asilomar Conference on Signals, Systems and Computers, Nov. 1993.

7. J.D. Northcutt and E.M. Kuerner, “System Support for Time-Critical applications,” Proc. NOSSDAV’ 91, Germany, pp. 242-254.

8. K. Jeffay, et al. “Adaptive, Best,-Effort Delivery of Digital Audio and Video Across Packet-Switched Networks,” Proc. NOSSDAV’ 92.

Table 1. Results for 160x120 resolution (Decoder)

No. of Bits of Lookup	PSNR (dB.)	Bandwidth as a function of frame rate (N) Kbps	CPU time per frame (ms)	Packing time per frame (ms)
4	31.63dB.	19.2 N	1.24 ms	0 ms
5	32.50 dB.	24 N	1.32 ms	0.52 ms
6	34 dB.	28.8 N	1.26 ms	0.80 ms
7	35.8 dB.	33.6 N	1.10 ms	1.09 ms
8	37.2 dB	38.4 N	1.18 ms	1.27 ms

Table 2. Results for 320x240 resolution (8 bit-lookup base)

No. of Bits of Lookup	PSNR (dB.)	Bandwidth as a function of frame rate (N) Kbps	CPU time per frame (ms)	Packing time per frame (ms)
2	33.72 dB.	48 N	6.01 ms	0.385 ms
4	35.0 dB.	52.8 N	6.04 ms	0.645 ms
5	35.65 dB.	62.4 N	6.05 ms	0.92 ms
6	36.26 dB.	67.2 N	6.08 ms	1.20 ms
7	36.9 dB.	72 N	6.04 ms	1.48 ms
8	37.5 dB.	76.8 N	6.09 ms	1.67 ms

Table 3. Results for 640x480 with 320x240 interpolated

No. of Bits of Lookup	PSNR (dB.)	Bandwidth as a function of frame rate (N) Kbps	CPU time per frame (ms)	Packing time per frame (ms)
2	33.2 dB	48 N	22.8 ms	0.385 ms
4	34 dB	52.8 N	22.87 ms	0.645 ms
5	34.34 dB	62.4 N	23.14 ms	0.92 ms
6	34.71 dB	67.2 N	22.93 ms	1.20 ms
7	35.07 dB	72 N	22.90 ms	1.48 ms
8	35.34 dB	76.8 N	22.95 ms	1.67 ms

Table 4. Results for 160x120 at the disk server

No. of Bits of Lookup	Bandwidth as a function of frame rate (N) Kbps	CPU time per frame (ms)	Seek-time (ms)	Avg. CPU Load
4	19.2 N	2.84 ms	16 ms	1%
5	24 N	3.67 ms	16 ms	1%
6	28.8 N	4.48 ms	14 ms	2%
7	33.6 N	4.92 ms	14 ms	2%
8	38.4 N	5.60 ms	16 ms	2%

Table 5. Results for 320x240 at the disk server

No. of Bits of Lookup	Bandwidth as a function of frame rate (N) Kbps	CPU time per frame (ms)	Seek-time (ms)	Avg. CPU Load
2	48 N	10.47 ms	18 ms	6%
4	52.8 N	11.02 ms	16 ms	6%
5	62.4 N	11.55 ms	18 ms	6%
6	67.2 N	12.29 ms	20 ms	7%
7	72 N	12.55 ms	20 ms	7%
8	76.8 N	12.73 ms	18 ms	7%