Adaptable Video Conferencing

Bharat Bhargava

1 Introduction

Video conferencing systems (VCS) have become practical in commercial and research institutions because the advances of technologies in networking and multimedia applications [4]. A video conferencing session involves multiple parties, possibly geographically interspersed, which exchange real-time video data. However, anomalies such as site failure and network partitioning affect the effectiveness and utilization of the communication capabilities. Video conferencing systems [4] lack the ability of dynamically adapting themselves to the variations in the system resources such as network bandwidths, CPU utilization, memory and disk storage. In VCS, changes in parameters such as frame sizes, codec schemes, color depths, and frame resolutions can only be made by users *interactively*. They cannot be made *automatically* based on the system measurements of currently available resources. We need to limit the users' burden in keeping the system running in the most suitable mode to current environment and make it possible to provide the best possible service based on the status of the system. Incorporating adaptability into a video conferencing systems minimizes the effects of the variations in system environments on the quality of video conference sessions.

This report is organized as follows. In section 2, we briefly discuss the concept of adaptability and the basic idea for achieving adaptability in a video conferencing system. Section 3 gives a description of common anomalies encountered in a distributed system. Section 4 first gives a overview of the NV video conferencing system, the testbed of our experiments. Then it described the extension and modification to NV and discuss some reconfiguration issues. Finally, a summary of experimental data analyses, observations, and discussions are presented.

2 Adaptability

Adaptability and reconfigurability are needed to deal with the performance and reliability requirements of a system. Current distributed systems provide a rigid choice of algorithms for application software implementation. The design decisions are based on criteria such as computational complexity, simulations under limited assumptions, and empirical evidence. The desired life cycle of a system is at least several years. During such time new applications surface and the technology advances, making earlier design choices less valid. In addition, during a small period of time (within a 24 hour period), a variety of load mixes, response time requirements and anomalies are encountered. An adaptable distributed system can meet the various application needs in the short-term, and take advantage of advances in technology over the years. Such a system will adapt to its environment during execution, and be reconfigurable for new applications.

Adaptability is particularly important for mission critical and distributed systems. Mission critical systems push current hardware and software technology to the limits, to meet extreme fault-tolerance or real-time performance requirements. Distributed systems must be adaptable to support heterogeneous hosts and to provide graceful degradation during anomalies. Computer networks support a larger and more diverse user community than centralized systems. Hosts of many different types running different software will be required to support this community. The underlying distributed system must be adaptable to a wide range of hardware and software. Furthermore, failures that would require complete shutdown of a centralized system can often be tolerated with only slight performance degradation in a distributed system. The distributed system software must be designed to be adaptable to a wide range of modes of operation, corresponding to the wide range of possible failure modes. For instance, [9] describes an algorithm for responding to failures by dynamically adjusting quorum assignments. As a failure continues, more and more quorum assignments are modified. Finally, when the failure is repaired those quorums that were changed can be brought back to their original assignments. By dynamically adapting to the failure the availability of data in the system is increased, at a cost that is only incurred during failure or recovery.

The principal advantages of adaptability are in the areas of reliability, performance, and software enhancement and maintenance. Adaptability provides for reliability through a design that is tolerant of site and communication failures. Adaptability improves performance because the system can adjust its transaction processing algorithms for optimum processing of the current mix of transactions. Adaptability simplifies software enhancement and maintenance through a design that is oriented from the start towards incorporating new ideas and approaches. Subsystems can be replaced without affecting other parts of the system, and the design of each subsystem supports the implementation of new algorithms. With maintenance costs climbing as high as 80% of life-cycle costs for many systems [7, 46], a design that supports future changes is becoming an essential part of software development.

2.1 Flavors of Adaptability.

We classify adaptability into four broad categories: structural static, structural dynamic, algorithmic and heterogeneous. Structural static adaptability consists of software engineering techniques for developing system software that can adapt to requirements changes over its life-cycle. *Structural static* techniques are categorized differently if they refer to layered software or unlayered software. Layered software supports *vertical adaptability*, the ability to replace a layer without affecting the other layers. Some layered software also supports *horizontal adaptability*, the ability to replace components within a layer[28]. Unlayered software can also take advantage of structural static adaptability. Fault-tolerance adaptability in hardware is classified into circuit level and module level. *Circuit level adaptability* in hardware corresponds to software techniques that check for errors within a software module, using redundancy [8] or correctness criteria [40]. *Module level adaptability* in hardware corresponds to techniques that isolate errors to within a single software module and allow the rest of the system to continue processing despite partial failures. Failure modes at the module level are often assumed to be *fail-stop*, which means that either the module does not produce a result or it produces the correct result. Module level and circuit level adaptability in software also refer to the implementation techniques that support replacement of software modules or algorithms, respectively.

Structural dynamic adaptability, usually called reconfiguration, is restructuring a running system in response to failures or performance requirements. Reconfiguration includes site failure protocols, network partitioning protocols, and other techniques for reorganizing the way the sites in a distributed system work together [39, chapter 5]. For instance, performance reconfiguration includes dynamically changing the communications structure of processors in a non-shared memory multiprocessor in response to changing tasks. Reconfiguration also has longer term benefits. New hardware can be easily integrated into a running system, and the software can be ported to new architectures with different interconnections between processors.

Algorithmic adaptability is a set of techniques for dynamically changing from the execution of one algorithm for a module to a different algorithm. For instance, a transaction system can change to a new concurrency controller, or a distributed system can change to a new site failure algorithm. Algorithmic adaptability can take place in one of three ways. *Temporal adaptability* refers to changes in algorithms over time. These methods generally have a brief conversion period after which operation continues with the new algorithm. *Per-transaction adaptability* consists of methods that allow each transaction to choose its own algorithm. Different transactions running at the same time may run different algorithms based on their requirements. *Spatial adaptability* is a variant of per-transaction adaptability in which transactions choose the algorithm based on properties of the data items they access. Spatial adaptability is an advantage in cases in which properties of different algorithms are desired for different data items.

Heterogeneity is the form of adaptability that deals with the issues involved in distributed computing using many different types of computing systems [39, chapter 6]. The simpler problems of heterogeneity include establishing physical connection between the machines, and resolving data type differences (e.g., size of integers, byte order). More difficult problems include getting diverse database systems to work together [26], and developing software that takes advantage of the particular strengths of each machine. Heterogeneity is an important problem for the future, as missions requiring several different types of computing engine will become more common [31]. Systems that can incorporate heterogeneous hardware and software components will also have the advantage of being able to incorporate new technologies more easily. Solving the problem of heterogeneity involves many of the problems in algorithmic and reconfiguration adaptability, and is aided over the long term by the software design techniques of structural static adaptability.

2.2 Techniques for Adaptability.

Adaptable software can be supported by an infrastructure of software tools. Support for modular design can make it easier to implement adaptability. The principle behind modular design is that abstract entities in the software design should be represented as physical entities in the implementation. For instance, separate activities should be represented as separate processes. In many operating systems, such as Unix, this means that the activities are represented as separate address spaces, each with a single thread of control. Shared resources are placed in one of the address spaces, and accessed from the other address spaces via messages or remote procedure calls. Since messages and remote procedure calls usually cost an order of magnitude more than local accesses, the performance of the entire system suffers. The problem is that the operating systems only offer a single process abstraction. More flexible process abstractions make sharing of resources such as memory and files efficient and easy. For instance, some operating systems support multiple threads of control within a single address space [23, 41].

Powerful remote communication primitives can also support adaptability. For instance, efficient multicast encourages the development of applications that can be spread among hosts in many different ways. Using logical multicast addresses the application does not have to worry about the location of the destination. Servers can relocate without informing their clients. Service requests directed to the logical address can be answered by any of a group of replicated servers [23].

New languages also offer support for adaptability. Object-oriented languages provide data abstraction to clearly define the interface to the programmer. Furthermore, the syntax and semantics of object references are the same whether the message is to an object in the same address space, in a different address space on the same computer, or even on a different computer. A smart compiler can exploit this flexibility by clustering objects into processes according to reference patterns, so most object invocations can be simple procedure calls [14]. In fact, the object can by dynamically moved as long as the interface is not affected.

Models of adaptability are another tool. In the paper [20] the sequencer model for subsystems of adaptable transaction systems was developed. The sequencer model formalizes three basic approaches to algorithmic adaptability: generic state, converting state, and suffix-sufficient state. With the model the problem of adaptability is reduced to mapping the subsystem to a sequencer, choosing one of the adaptability approaches, and implementing it. Software engineers have studied design techniques that enhance structural static adaptability. Models for other types of adaptability are needed.

2.3 Using Adaptability.

Incorporating adaptability into a system requires an understanding of how and when adaptability will be used. Structural static adaptability is used every time the software is changed. Any software that will include maintenance in its life-cycle should incorporate techniques to make maintenance easier. Reconfiguration adaptability is necessary in systems for which availability is important or reconfiguration is common. Reconfiguration adaptability may also be necessary for mission critical systems that must use different configurations to respond to different situations. Algorithmic adaptability is most important in systems that are pushing hardware performance to the limit, or that require very high levels of fault tolerance. Algorithmic adaptability is useful for responding to environmental changes that are predictable, or that last long enough to amortize the cost of the adaptation. Finally, support for heterogeneity is important in any system that has a long life-cycle or that will run on a large network of computers. In choosing when adaptability can be applied there is a tradeoff between the costs and benefits. Experimental work is needed to evaluate adaptability in practical environments.

2.4 Adaptability in Video Conferencing System

A video conferencing system should provided some policies and mechanisms to make it adaptable to the anomalies based on the available resources. The advantages of the adaptability schemes for VC system includes:

- **Heterogeneity:** A VC system will adapt to heterogeneous environments. That is, a video conference session can be held on different hardware platforms and different networks.
- **Scalability:** A VC system will adapt itself as more users, more sites join a video conference in progress.
- **Anomaly Management:** A VC system will adapt to anomalies and degrade gracefully when available resources decrease or become unavailable.
- **Resource Management:** A VC system can make efficient use of resources like storage, CPU time, and communication bandwidth.

The basic idea for achieving adaptability for video conferencing system is to trade some aspects of video quality for others. For example, the frame rate decreases as the available network bandwidth drops. Since the smoothness of a video session is sometimes more important than any other aspects of video quality, we may have to maintain a reasonable frame rate during a video conference session even though the network performance becomes degraded. To achieve this, we have to sacrifice some aspects of video quality, such as color or resolution of video frames.

Adaptability can be achieved ether by the user intervention or by the system itself. If user intervention is required, the system accepts inputs from the user and change the level of service or some system parameters according the the new specification. If adaptability is to be done by the system automatically, the distributed control system periodically measures the available resources and supply these parameters to the video conferencing system. Based upon the current parameters, the video conferencing system reconfigure itself in a user-transparent way to provide the best possible service based on some user-specified criteria that must be satisfied.

3 Anomalies in a Distributed System

Video conferencing tools are being used on top of distributed systems. Therefore, the anomalies in distributed systems will affect the effectiveness and utilization of a video conferencing tool. Typical distributed system anomalies includes:

- **System resources limits:**
  System resources are usually limited and change dynamically. Common limited resources includes available network bandwidth, CPU time and disk storages.

  Network bandwidth in a distributed system is available in limited quantities and varies with network traffic. For example, the bandwidth of a typical Ethernet is 10 Mbps. Available network bandwidth may decrease drastically during a short period of time due to many users who send large objects (packets) simultaneously. This is the typical case for video conferencing sessions. When the available bandwidth drops sharply, the video conferencing system can change the level of service by changing color or frame resolution to reduce the objects' size.

  Depending on the load on the system, CPU time available for the video conferencing processes changes. The system may choose to avoid compressing the data before sending to decrease the CPU utilization if the available bandwidth is large enough.

  Availability of storage space and memory varies from machine to machine in a distributed system. For example, a server machine in an organization may have several gigabytes of disk space and over 48 megabytes of memory, while a personal computer participating as a client may have only 500 megabytes of disk space and 4 megabytes of memory. In face of heterogeneous environment like this, the system may choose to provide different kinds of service for each type of machine by negotiating before connecting. For example, the server may decide to reduce the frame rate or frame resolution if the participating site is a PC, while maintaining high quality video frames for participating hi-end workstations.

- **Failures:**
  A failure is a deviation of a system from its normal operation due to some errors or faults. Various kinds of failures can occur during a video conferencing session. These failures ranges from hardware failures, software failures to network failures and many others. Among them, the most relevant failures to a video conferencing system are:

  - Site failures: the participating site may be down for some time and re-join the session later. This could be caused by various kinds of errors from hardware, operating system, software, memory or disk. To deal with site failure, a video conferencing system may record the session in some permanent storage and the failed site may retrieve them later when it back to normal operation. To prevent site failure caused by disk error, a distributed system may use redundancy storages to store duplicate copy of data.
  - Network failures: the network may be disconnected for a period of time, or the network may be congested due to large volume of traffics. In order to deal with network failure, the packets may be re-routed through other connections, or the data to be sent can be stored in some temporary storage and retransmited when the network become available.
  - System failures: This refer to a wide range of system limit violations. For example, the hardware may be malfunction for some period of time. There may be too many processes running on a system, therefore no additional process can be created when needed. The disk space may be full, therefore no temporary file can be created and the normal processing of the system can't be continued. To deal with system failures, the application may go into a recovery mode of running and wait for the system to back to its normal state, then it can keep on running on its normal mode.

- **Number of participant varies:**
  The number of users participating in a video conferencing session varies from time to time. This will change the demands placed on the video conferencing system. If the load of the system is too heavy during some session, the system may limit the number of users in order to provide reasonable service to each participant. It can also degrade the level of service for some users in order to maintain the system availability to all the users.

To deal with anomalies, a system can just detect and report them, or it can tolerate them. Furthermore, a system can be designed to fix the anomalies. A system must be able to detect the anomalies and avoid getting into an inconsistent state. There are two different kinds of anomalies. Each can be dealt with a different strategy. Anomalies can pop up fast (suddenly) or grow slowly. For example, a power failure can occur suddenly and cause the system to stop immediately. If anomalies occur suddenly, the system may not prepare enough to recover from it and may stop working immediately and does not cause further damages to the system. This is called a fail-stop mode. However, sometimes the failure of the system can cause cascade damages to the system. The distributed system should be prepared to deal with sudden anomalies and guarantee the anomalies will not cause a lot of inconsistency to the system. For example, database log constantly record the system status in the permanent storage. If a sudden failure occurs, the system can recover the database back to some old but consistent state. Although some transactions may be redo or undo, the information contained in the system log can guarantee that no further damage will occur to the system. A video storage system can record all frames for a session and then make them available to play again. If anomalies grow slowly, the distributed system can prepare more for recovery. It can be designed to tolerate them, maybe by degrading the service gracefully, or it can try to fix the anomalies and back to normal processing. If the system want to tolerate the anomalies, it may substitute or reconfigure one of its components. However, the substituted component may be inefficient, it may be reliable only under limited circumstances (inputs), or it may be operational only for a limited time. For example, if a direct link between two sites was disconnected, the system may decide to re-route the data through some indirect route, with the penalty of additional delay for the data. Of course, there are always some overheads that exist if a distributed system must prepare to deal with anomalies. For example, checkpoint is used to help recover processing back to a consistent state in case of some failures, but there are some overhead associated with checkpoint due to periodically recording of the system status.

Adaptability allows a distributed system to accommodate anomalies and reconfigure itself to provide service based on the available resources.

4 The NV software

Network Video (NV) [1], a popular video conferencing tool developed at XEROX PARC was the tool we chose as our testbed to implement adaptability features. We have extended NV to provide adaptability to network characteristics and reconfigure itself different parameters during anomalies such as the ones listed above. Being hardware-independent, we could compare the overheads of NV involved in the software implementation of our algorithms without having to exclude the performance benefits due to the hardware realization of certain codec schemes.

4.1 Structure of NV

NV transmits and receives video data across the Internet using RTP built on top of UDP. It is designed to run over a wide range of network bandwidths – it can support sending slow frame rate video over a slow modem line, while also allowing higher quality video to be sent over a high speed LAN [1]. NV uses a video compression scheme designed to perform at a reasonable speed when implemented in software. It takes advantage of the similarity between consecutive frames in a typical video stream and the similarity between neighboring pixels in any given region of the frame. This block-based compression scheme takes only tens of milliseconds on a workstation to compute [1]. The stationary blocks are periodically transmitted to improve image quality. Thus packet losses are made up by retransmission triggered by motion.

NV has three main modules: the *grabber* module, the *encoder* module and the *sender* module. The grabber module grabs frames using the platform specific video capture board. It performs the analog to digital conversion in hardware. The encoder module does the lossy compression. Human eyes are more sensitive to luminance than chrominance and by converting the RGB model to YUV (the process of subsampling), the encoder module makes it possible the reduce chrominance information and retain the luminance information. The encoder module also executes the block-based conditional replenishment algorithm. Haar transformation is applied to each block to pick up the high frequency components to help in getting better looking text. The encoding process is finished with dead-zone quantization where a single threshold value is used to eliminate low energy terms and finally run-length coding.

4.2 Reconfiguration of NV

The adaptability features for NV include *color depth*, *frame resolution*, *frame size* and *codec scheme*. In order to incorporate adaptability into NV, there are some important aspects which must be considered. These aspects includes:

4.2.1 Inputs

In order to notify NV about the changes of the system environment, there should be some inputs from outside of NV. Input parameters for adaptability which should be monitored from NV application includes number of users, delay/jitter, frame rate, color, frame size and frame resolution. Input parameters that are available in DSC includes CPU utilization, available network bandwidth, storage requirements. The inputs can come from user, which means the parameters can be selected and modified interactively by the user when the service is unsatisfiable, or it can come from the distributed system. That is, the inputs come automatically from the distributed system based on current parameters of the environment like available bandwidth and CPU utilization, etc.

If the input is from user, it can be specified either by selecting the level of service or by specifying actions which must be done by NV such as *compress color depth* or *reduce resolution*. It can also be some direct change to any parameters mentioned in the previous paragraph such as color or frame size.

If the input is from the distributed system, the distributed system must periodically measures the available resources (including network bandwidth and CPU utilization). Based on the currently available resources, the distributed system calculate some input parameters for NV. The input parameters for NV are the results of some decision made by the distributed system (usually in the form of some object function evaluation) in terms of one or more QoS parameters and application attributes timeliness, precision, and accuracy. Upon receipt of the new parameters, NV reconfigures itself to run under new level of service.

When anomalies are detected and new available resources are measured, DSC must make decisions on what action should be taken by NV and notify NV its decision. The decision of which actions to take is described in the section *Reconfiguration Policy* and how the parameters and behaviors of NV will be changed when the policy is chosen is described in the section *Reconfiguration Mechanisms*.

4.2.2 Reconfiguration Policy

Reconfiguration policy is a decision derived from a set of decision rules. Reconfiguration policy is used to guide the choice among several alternatives when a reconfiguration is possible and needed. In a reconfiguration policy, there are several sets of mutual exclusive alternatives which have to be traded off. These trade-off sets includes:

- Communication/Compression:
  The trade off between communication and compression depends on the available network bandwidth and CPU utilization. Figure 1 list the CPU utilization under various level of service.

  If the data are compressed before sending, it takes less bandwidth to send the data. However, it takes additional CPU time to compress the data at the sender side and takes additional CPU time to decompress the data at the receiver side. On the other hand, if the data are send without compression, it takes more bandwidth to send the data but less CPU time for both the sender and receiver to process the data. Figure 5 and 6 show that the percentage of time in encoding and decoding over total processing time decreases as the compression ratio increases from 1:1 to 1:16. Therefore the decision of whether to compress the data should be made based on the available network bandwidth and CPU utilization.

- Timeliness/Accuracy/Precision:
  Timeliness is defined as "when an event is to occur". Maintaining it means meeting a deadline. Accuracy is defined as "the degree to which the output conforms to the semantics and contexts of the applications". Maintaining it means guaranteeing the correctness of the data. For example, lossy compression algorithms cause the loss of accuracy. Precision is defined as "the quantity of information provided or processed". Maintaining it means maintaining the amount of data being processed or transmitted over the network. For example, the number of frames per session, the number of pixels per frame, and the number of bits per pixel are parameters used to describe the precision of a video conferencing session.

Figure 1: CPU Utilization under Various Levels of Service

Level	Configuration	CPU Utilization
1	(G, 128, S)	67 - 72
2	(G, 256, S)	76 - 81
3	(G, 512, S)	78 - 83
4	(C, 128, S)	64 - 69
5	(C, 256, S)	74 - 79
6	(C, 512, S)	77 - 82
7	(G, 128, M)	80 - 85
8	(G, 256, M)	86 - 90
9	(G, 512, M)	89 - 93
10	(C, 128, M)	78 - 84
11	(C, 256, M)	86 - 89
12	(C, 512, M)	91 - 94
13	(G, 128, L)	92 - 97
14	(G, 256, L)	95 - 98
15	(G, 512, L)	95 - 98
16	(C, 128, L)	95 - 97
17	(C, 256, L)	95 - 97
18	(C, 512, L)	95 - 97

TAP cannot be maintained at the highest level simultaneously during anomalies. We must trade among these attribute values. The policy to trade among these attributes can be described as follows:

- Maintaining Timeliness when Bandwidth Decreases
  - Reduce frame size (The accuracy is maintained unless the frame size is below a certain value).
  - Reduce frame resolution (Both accuracy and precision are reduced).
  - Dither color frame to black and white.
  - Compress color depth.
  - Switch to a codec scheme that has a higher compression ratio (Side effect: CPU utilization increases. This can be compensated by frame resizing and resolution reduction).
- Maintaining Accuracy when Bandwidth Decreases
  - Switch to a lossless codec scheme with reduced frame size.
  - Dither color frame to black and white.
  - Compress color depth (compress Y and UV no more than 2 bits each).
  - Do not use lossy codec schemes.
  - Do not reduce frame size or resolution by a big factor.
- Maintaining Timeliness when CPU Utilization Increases
  - Switch to a codec scheme that requires less computation (usually with lower compression ratio).
  - Reduce frame size.
  - Dither color frame to black and white.
  - Do not compress color depth.
  - Do not reduce frame resolution.
- Maintaining Accuracy when CPU Utilization Increases
  - Switch to a lossless codec scheme
  - Reduce frame size.
  - Dither color frame to black and white.
  - Do not compress color depth.
  - Do not reduce frame resolution.
  - Do not use lossy codec schemes.

4.2.3 Reconfiguration Mechanisms

Reconfiguration mechanisms refers to the implementation of the reconfiguration policy that has already been decided. It is called the *infrastructure of the adaptability* in a system. For our infrastructure, we extend NV by adding new modules and modifying the three modules that exist in the original design. The extended architecture is depicted in figure 2. The new modules added to NV are **Network-Probe Module**, **Admission Control Module**, and **VC adaptability Collaborator Module**. Network-probe module is used to constantly monitor the network. It is integrated with WANCE (Wide Area Network Communication Emulation) tool to track round trip time and routing information of the packets. Admission control module is used to restrict number of users if there are insufficient resources available. VC adaptability Collaborator Module includes three new modules, namely **Recorder Module**, **Summarization Module** and **White Board Module**. Recorder Module is used to store video data frames for future retrieval, retransmission and reference. Summarization Module is used to perform user-aided extraction of significant frames in a session based on frame contents. White Board Module provides a way to write on, send text, graphs and other material during a conference. There will be detailed explanations for each of the new module in this section later.

In addition to the new module, there are some changes in the three original modules. The modifications are:

- **Grabber Module**
  - Change sampling rate, which is the frequency of grabbing frames in a stream of input. For example, grab alternate frames instead of all of them.
  - Reduce load on the sender machine - saves computation in the encoder module.
  - Reduced load will help sender receive better (packets will not be lost).
- **Encoder module**
  - Incorporate switching between Haar and DCT transformations depending on the application.
    - Haar is good for video frames with high frequency components (like edges).
    - DCT is good for other kind of frames like movie frames.
  - Add the ability to encode different levels of color:
    Full color: 24 bits per pixel (YUV)
    Greyscale: 8 bits per pixel
    - Extract the 16 UV bits.
    - Perform 4:1 reduction.
    - Merge the new 4 UV bits with the original 8 Y bits to get a total of 12 bits.

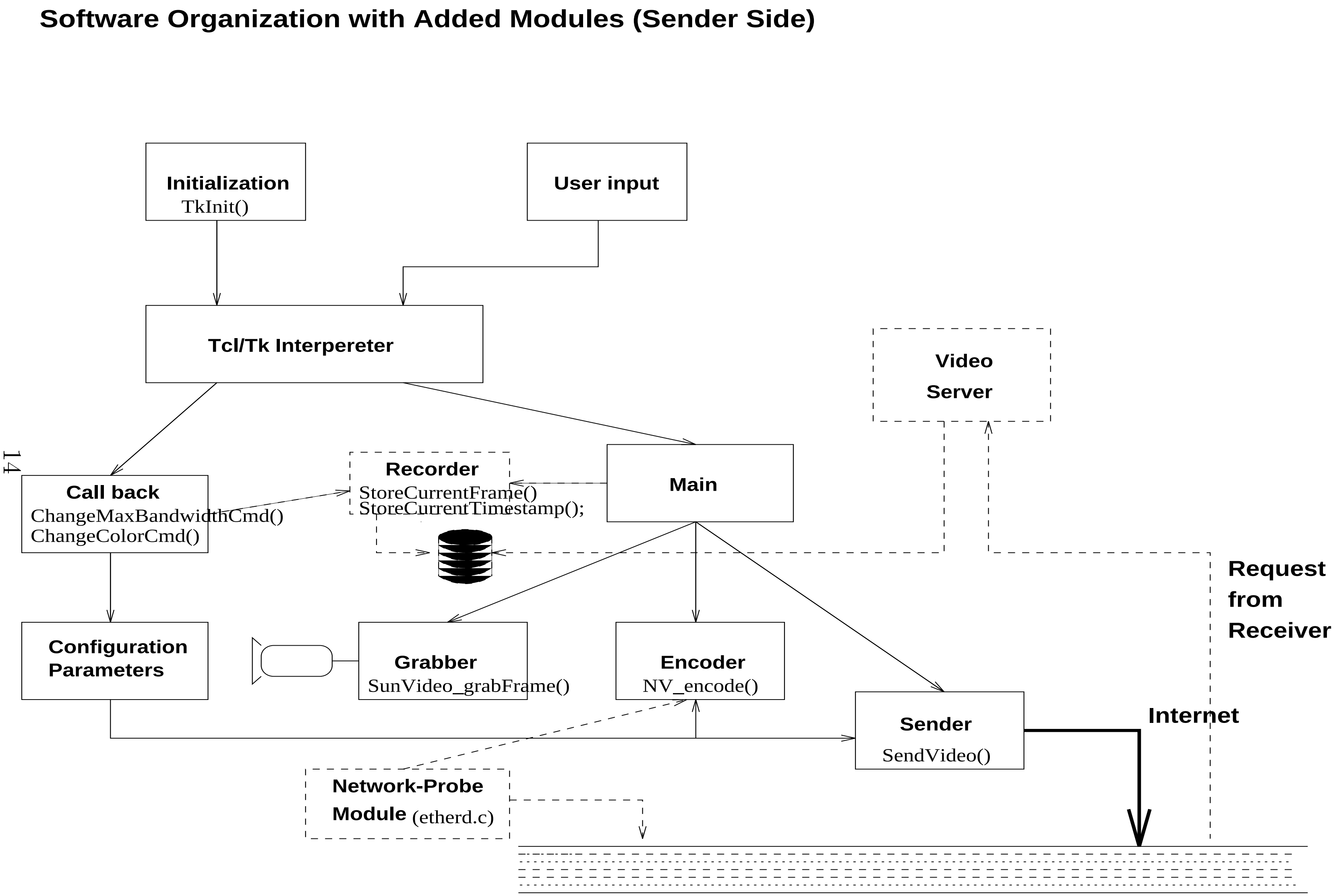

Figure 2: Extended architecture of NV

(Similar to the dithering procedure VidColor_MSB8bit())

– Change resolution of frames Compute the value of a pixel as an average of 4 pixels.

$$\boxed{\begin{matrix} X_1 & X_2 & X_3 & X_4 \\ X_5 & X_6 & X_7 & X_8 \\ X_9 & X_{10} & X_{11} & X_{12} \\ X_{13} & X_{14} & X_{15} & X_{16} \end{matrix}} \xrightarrow{4:1} \boxed{\begin{matrix} Y_1 & Y_2 \\ Y_3 & Y_4 \end{matrix}} \longrightarrow \boxed{\begin{matrix} Y_1 & Y_1 & Y_2 & Y_2 \\ Y_1 & Y_1 & Y_2 & Y_2 \\ Y_3 & Y_3 & Y_4 & Y_4 \\ Y_3 & Y_3 & Y_4 & Y_4 \end{matrix}}$$

Can be coded in SunVideo_Reconfig()

- **Sender Module**
  - Switch protocols depending on current network conditions to achieve reliable transmission, better flow control and better jitter control
  - Prioritize sessions
    * Add callback function SetPriority() so that a user can specify priority at the beginning of a session (static)
    * Add callback function ChangePriority() so that a user can modify a priority when a session is in progress (dynamic)
    * When network is congested, lower priority sessions are stored and not transmitted
  - Admission Control - restrict the number of users in a session.

The network-probe module monitors the network constantly. It measures network traffic, detects congestion, and calculates available bandwidth. It interfaces with sender and encoder modules through shared memory. The variable bandwidth can be accessed by all three (grabber, encoder, sender) modules. There are two domains in monitoring the Internet. The network-probe module can either monitor the local network till the gateway, or it can go beyond the gateway. However, statistics gathered beyond the gateway might not be accurate since we don't know which route the packets are going to take. In our implementation, we integrate the network-probe module with WANCE (Wide Area Network Communication Emulation) tool to get a better environment for our experiment.

The admission control module restricts the number of participants per session if sufficient resources are not available. It stores the session and retrieves it for playing back later for users not able to join a session. It also prioritizes sessions, media, recipients, and messages. It will cut conversation time or close the conversation for sessions with lower priorities and send the most important messages first if the conversation time is limited. It closes the less important medium to save bandwidth when the available bandwidth decreases. In case of site failures, network congestion or extreme low bandwidth, the admission control module uses VC collaborator to record video session for later playback, retransmission and retrieval.

The recorder module record video frames in the video-stream database at both the sender and receiver machines. The functionalities includes:

- Record frames every fixed interval (automatic, inflexible).
- Record the changed part (difference, mostly foreground).
- Record frames in variable rates (based on relative importance).
- Record frames based on user-defined criteria (e.g., some particular time intervals), or frame features (e.g., some particular scenes).

The summarization module is to perform user-aided extraction of significant frames to organize frames by theme. It includes off-line processing (since it is time-consuming and hard to achieve real-time) for typical session. It also includes on-line, user-assist processing for slowly evolving scenes. However, to effectively summarize a session, some CAS (Computer Aided Summarization) tools are needed. In addition to video, audio provides more information and can be combined into the summarization module for processing.

White board module provide a user interface to NV. The motivation to add a white board module is to let the user see the speaker and other information such as slides, graphs, images, maps at the same time. However, using the same window for speaker and information would generate a lot of traffic in NV because of background change. In our design the sender and receiver can interactively point to and modify objects on the white board. In our implementation, user inputs will be parsed using the Tcl/Tk interface and grabbed from the screen using the X11_Grab module. There is also an interface with the VC adaptability collaborator in our implementations.

4.2.4 Experiments and Results

1. Experimental Setup:

   We chose Network Video (NV) [1], with enhancements to incorporate adaptability and recording feature, as our testbed to conduct performance studies.

   The platforms for the experiments include a Sun Sparc 10 station and a Sun Sparc 5 station connected in a LAN environment, and a couple of video cameras. The Sun workstations are running Solaris 2.3 operating system.

2. Observations and Discussions

   - Experiments on Color Depth Compression

     Two color models are often used in practice: RGB and YUV [3]. Y stands for the luminance component of a pixel; U and V represents the chrominance components of a pixel. Many systems use YUV model because of its ease of adjustment for colors.

A

Y \ UV	0	1	2	4
0	original	good	good	noisy
1	snowy	good	good	noisy
2	less snowy	snowy	good	noisy
4	with grid	with grid	less snowy	noisy

C

B

Figure 3: Image Qualities for Different Compression Factors for Y and UV Components

For example, in YUV model, UV components can be changed easily and separately without disturbing Y components, and vice versa. In NV system, frames represented in RGB space are first captured and then converted into those in YUV space. This process is called subsampling. Since human eyes are less sensitive to chrominance than to luminance, we can reduce the number of bits in encoding U and V components while retaining the number of bits in encoding Y components, and still guarantee relatively good quality in decoded color frames. Nevertheless, subsampling is lossy.

Although we can achieve different levels of colors by using different number of bits to encode UV components, it may still be needed to reduce the number of bits in encoding UV when the bandwidth is too small and the number of bits in encoding Y cannot be reduced any further.

In NV system, Y and UV components of a pixel are each 8 bits long. Color depth compression is achieved by using fewer bits to represent Y and UV components, respectively. We have experimented with various configurations set up by separately compressing Y and UV components each by 1, 2, and 4 bits. For example, compressing UV components by 2 bits means they are represented with only 6 bits. Figure 3 shows the effect of reducing the number of bits in UV and Y components of a pixel on the quality of a video frame. We divide figure 3 into three regions: A, B, and C, as described below:

(a) **Region A:** Very good image quality can be achieved when either Y components alone or both Y and UV components are compressed by a small number of bits

Figure 4: Frame Rates for Resized Frames

(up to 2 bits). Thus, up to 4 bits of combined Y and UV component compression can be achieved when a very good image quality is being maintained.

(b) **Region B:** Human eyes can tolerate (accept) image quality in this region when the number of bits compressed for Y components is less than 4 bits. This can be attributed to more compression in UV components than in Y components. Thus, more bits in encoding UV components *can* be reduced than those in encoding Y components as long as the compression for Y components does not reach a threshold value (4 bits).

(c) **Region C:** The image quality in this region drops dramatically (though the image is still recognizable) when the number of bits compressed for Y components reaches a threshold value (4 bits in this case). Thus, we *have to* compress more bits for UV components if the bandwidth becomes too small, because the number of bits in encoding Y cannot be reduced any further.

Therefore, we can draw some conclusions as follows:

(a) We stay in Region A and obtain really good image quality if the combined number of bits of compression for Y and UV components of a pixel is not expected to exceed 4 bits.

(b) We enter Region B where we can get not-so-bad image quality when the combined number of bits of compression for Y and UV components of a pixel is greater than 4 bits, but is no larger than 6 bits.

(c) We go to Region C when we need a higher compression rate in combined number of bits to encode Y and UV components of a pixel. But image quality is degraded substantially. Due to the relative importance of luminance components, we should *not* compress Y for more than 4 bits if we want to maintain a recognizable image quality.

- Experiments on Resolution Reduction and Frame Resizing

  To see how the frame resizing can be used for adaptability for video conferencing, we conducted some experiments to measure the frame rates for frames of different sizes (Figure 4). For a particular frame size (e.g., 640 x 480), as the available network bandwidth decreases, the corresponding frame rate also decreases. This will result in loss of continuity and smoothness of video presentation at the sender as well as remote sides. In such a situation, the VC system would adapt by changing to smaller frame size to maintain (or even improve) the original frame rate. At present, the system supports only four discrete levels of frame sizes. Thus, the frame rate may be changed (improved) when the system changes to a level of service with less bandwidth requirement. In the future, we plan to provide more levels of frame sizes, which will allow the system to adhere (very closely) to the current operating frame rate while reducing the network bandwidth requirements at the same time.

To implement the dynamic frame resizing and resolution reduction, we manipulated data before it is encoded and after it is decoded. Though it takes extra CPU time, our results (figure 5, figure 6, figure 7, and figure 8) show that this overhead is tolerable. We computed the average time percentages for encoding and decoding in processing one frame. In frame resizing, The combined time percentages for encoding and decoding are more than 50% of the overall processing time of one frame, which means that encoding and decoding are the most expensive parts in video conferencing. When the compression factor value become 2 or 4, the combined time percentages for encoding and decoding drop dramatically; they are no longer the most expensive parts of processing in video conferencing. Instead, the video transmission becomes the most expensive part of processing in video conferencing. Similarly, in resolution reduction, the time percentage for encoding decreases when the frame size is decreased. However, the time percentage for decoding when compression factor is equal to 2 or 4 increases compared to that when compression factor is 1. This is due to the extra computation overhead involved in the resuming the frame size to the original one. But the combined time percentages of encoding and decoding over overall processing time are only slightly bigger than that for original NV.

- Experiments on Codec Schemes

  The original NV software supports three codec schemes, namely, native NV scheme, CellB scheme invented at Sun Microsystem, Inc., and CU-SeeMe scheme. Our modified version of NV also supports a scheme that replaces the Harr transform in the native NV scheme with Discrete Cosine transform (DCT) algorithm. DCT is the core transform used in many codec schemes, including JPEG [5] and MPEG [2]. We have implemented DCT algorithm in NV and studied the performance of the four codec schemes in the context of NV.

  The native NV compression scheme is a sequence of Harr transform, quantization, and run-length coding. The uncompression process is the inverse. We replace Harr transform with DCT algorithm in the compression process and replace Harr inverse transform with inverse DCT in the uncompression process.

  The importance of the incoming video data from the camera has been ignored to certain degree in the performance study of video conferencing softwares. In this paper, we investigate this in some detail. We found that the CPU utilization and frame rate

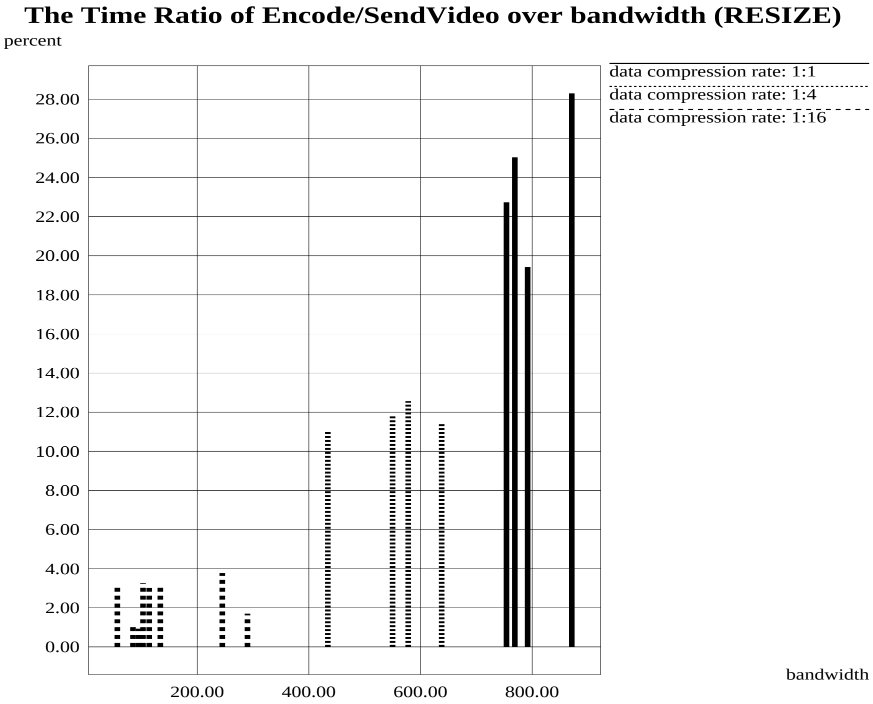

Figure 5: Percentage of Time in Encoding over Total Processing Time for Resizing

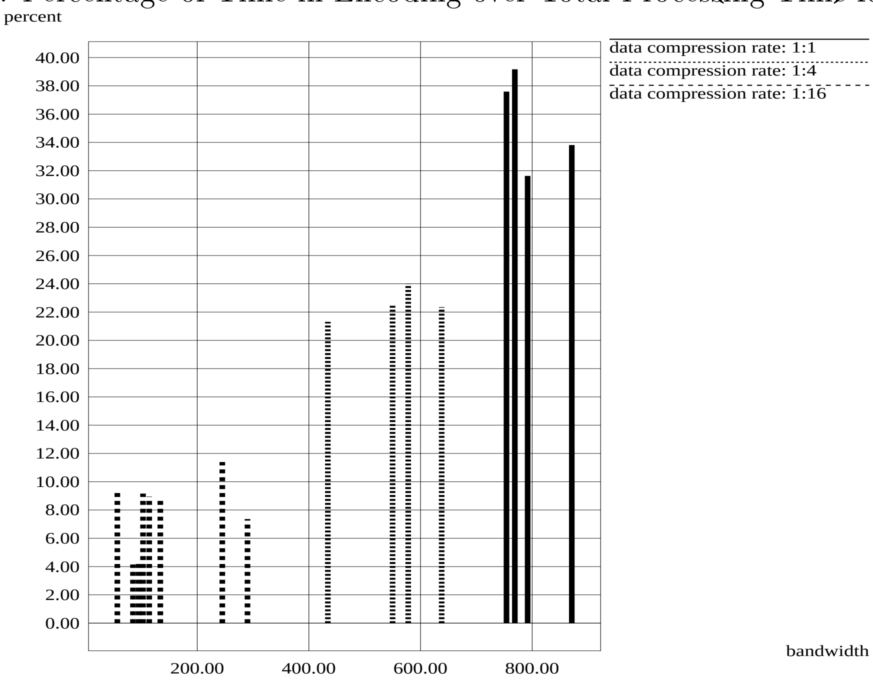

Figure 6: Percentage of Time in Decoding over Total Processing Time for Resizing

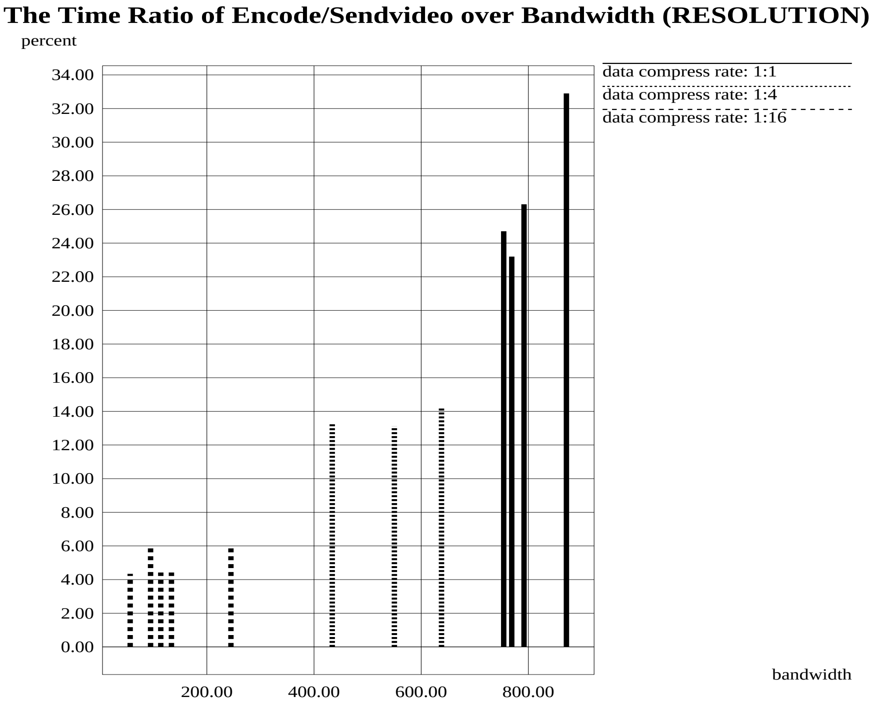

Figure 7: Percentage of Time in Encoding over Total Processing Time for Resolution Reduction

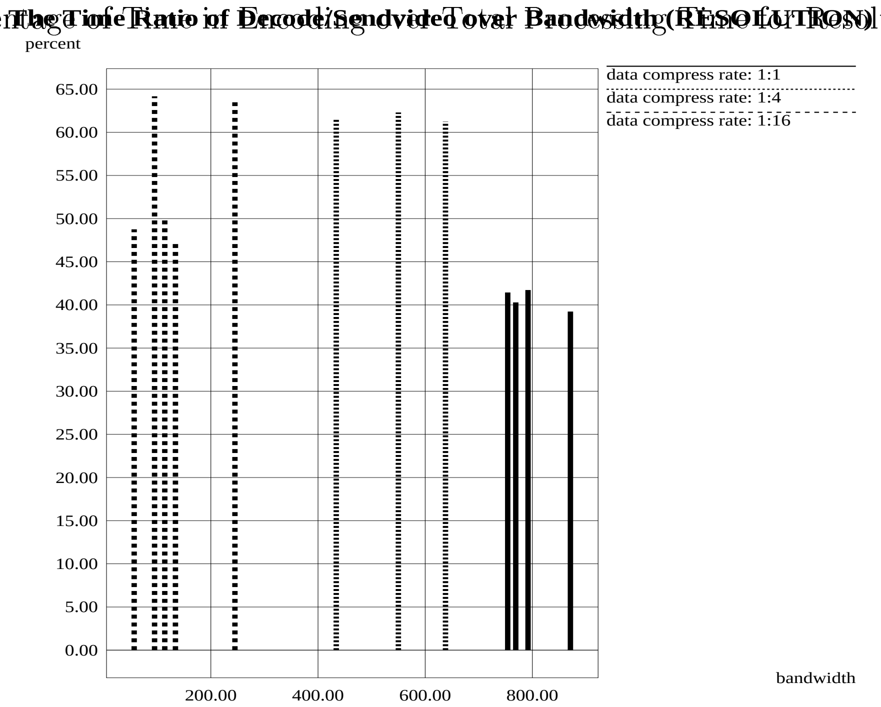

Figure 8: Percentage of Time in Decoding over Total Processing Time for Resolution Reduction

Coding Scheme	Size	GreyScale			FullColor		
		Small	Medium	Large	Small	Medium	Large
Native NV Scheme		[62,67]	[72,80]	[99,103]	[61,66]	[71,77]	[100,103]
DCT Scheme		[63,67]	[74,75]	[103,108]	[60,63]	[70,73]	[101,105]
CellB Scheme		[4.6,6.5]	[4.9,6.5]	[4.3,7.6]	[5.5 ,7.4]	[6.4,7.4]	[6.5,9.4]
CU-SeeMe Scheme		[66,70]	[73,76]	[99,101]	Color Mode not supported		

Table 1: CPU Utilization in various Codec Schemes supported by NV

also depends on the amount of motion in each frame and its frequency distribution. The latter in turn depends on the content of a frame. For example, a textual frame usually has more high frequency components whereas a continuously toned frame has more low frequency components.

To measure the amount of motion in a precise way, we define *momentum* of a frame to be the number of blocks whose contents have been changed. We define *relative momentum* of a frame to be the percentage of the number of blocks whose contents have been changed over the total number of blocks in that frame. Our hypothesis was that the processing time for video data heavily influenced by the momentum. Our experiments support our assertion positively.

We have done a series of experiments to study the performance of NV system under various configurations. The configurable parameters include codec schemes (native NV scheme, CellB scheme, DCT scheme, and CU-SeeMe scheme), frame sizes (160 x 120 (small), 320 x 240 (medium), or 640 x 480 (large)), and display modes (color mode or greyscale mode). We measure the CPU utilization of NV under various configurations, as shown in table 1.

Note that in many cases displaying frames in greyscale mode takes more CPU time than in color mode. This appears to be a contradiction to the fact that displaying frames in color mode is more expensive than in greyscale mode. However, it implies that the frame rate in greyscale mode is higher than that in color mode. Table 2 shows the frame rates under the same configurations.

Figure 9 and 10 show how the distribution of the time spent in forward Harr transform and DCT for a frame is dependent on the relative momentum of that frame. Figure 11 and 12 show that for corresponding reverse transforms, the bigger the relative momentum, the longer a transform lasts.

We have compared the performance between NV scheme and DCT scheme. Figure 13 and 14 show that both forward and inverse transform of DCT takes more time

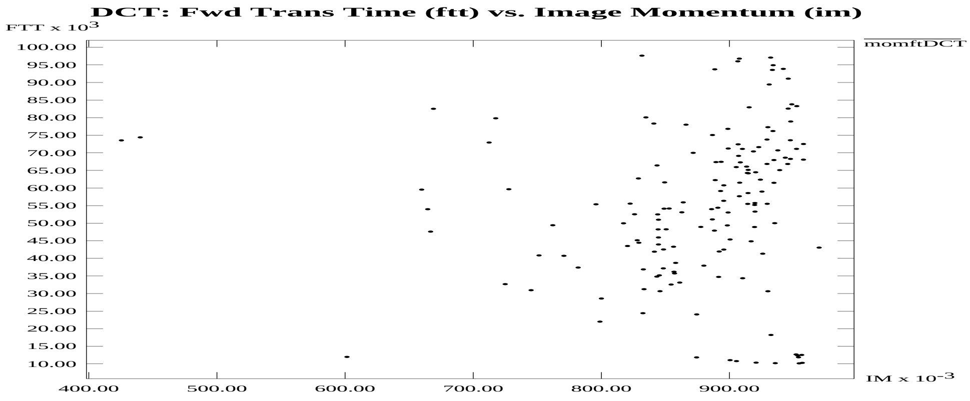

Figure 9: Impact of Momentum on DCT Forward Transform Time

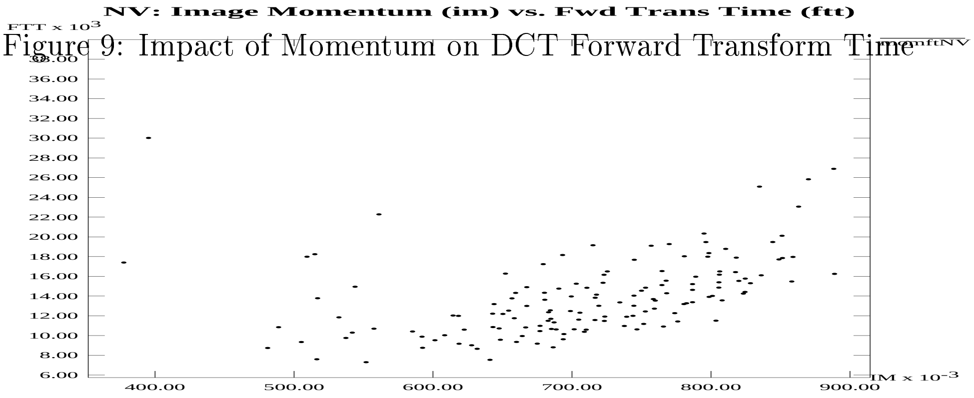

Figure 10: Impact of Momentum on NV Forward Transform Time

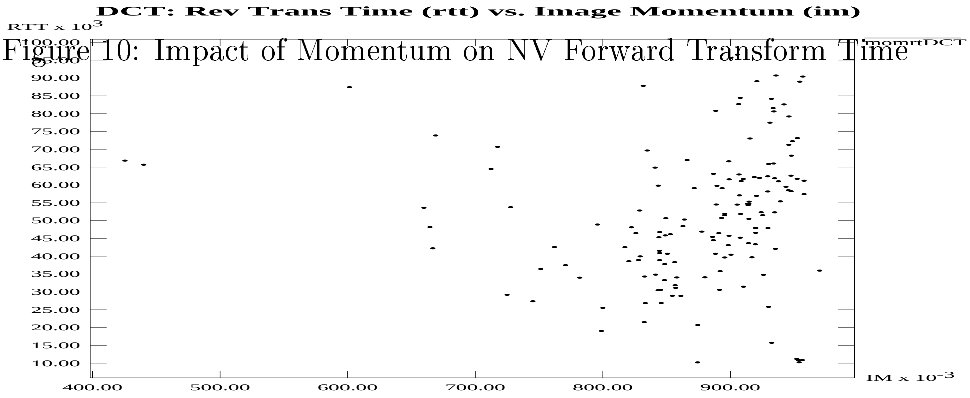

Figure 11: Impact of Momentum on DCT Reverse Transform Time

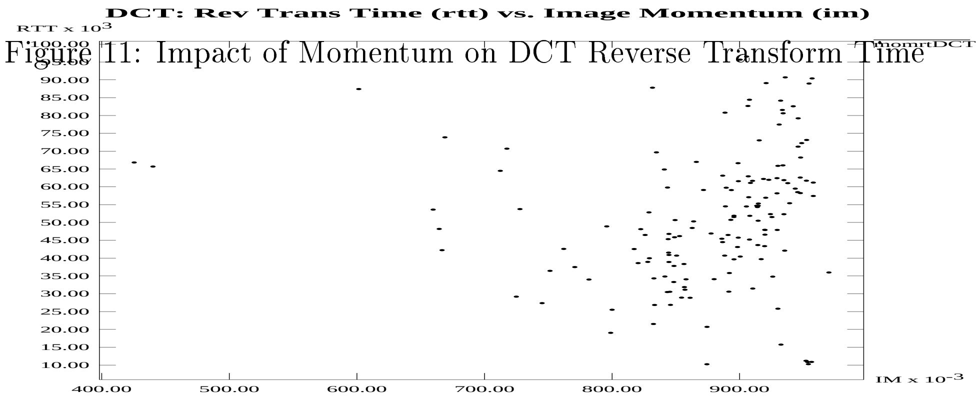

Figure 12: Impact of Momentum on NV Reverse Transform Time

Coding Scheme / Size	GreyScale			FullColor		
	Small	Medium	Large	Small	Medium	Large
Native NV Scheme	1.9	0.5	0.2	1.6	0.4	0.2
DCT Scheme	1.9	0.8	0.3	0.7	0.2	0.1
CellB Scheme	6.0	1.5	0.3	6.0	1.6	0.3
CU-SeeMe Scheme	5.9	1.0	0.8	Color Mode not supported		

Table 2: Frame Rate obtained using various Codec Schemes supported by NV

Figure 13: Time Spent in Forward Transform: NV vs DCT

Figure 14: Time Spent in Reverse Transform: NV vs DCT

Figure 15: Compression Ratio: NV vs DCT

than those of NV. This is expected since cosine functions in DCT are transcendent, and Harr functions are elementary, which implies that DCT is more computationally expensive than Harr transform. But the result about the compression ratio is not in the realm of our expectation. Figure 15 shows that the compression ratio of NV is about twice as that of DCT. This test was done for continuously tuned frames. DCT should have better compression ratio than Harr transform for such frames.

- Experiments on Session Recording

  Session recording involves both the computation overheads and secondary storage requirements. One important parameter that we consider is the *recording frequency*, the frequency at which the video frames are stored. This must be differentiated with the *display frequency*, the frequency at which the video frames are updated on a user terminal. The latter determines the continuity and smoothness of the video session presentation. Storing every video frame generated during a video session is sometimes not necessary because many consecutive frames may have almost the same contents (for example, because of lack of scene changes) and because it would consume a lot of disk space. On the other hand, recording frames at a very low recording frequency may hurt the smoothness of video sessions and degrade the video quality during replay.

  Recording video frames imposes computational overheads which can affect the rate at which video frames are generated for display (i.e. the display frequency). Since the system records the VC sessions only when an anomaly occurs, for example, when

Figure 16: Effect of Recording Overhead on Frame Rates

Figure 17: Effect of Changing Recording Frequencies on Sizes of Recordings

the available network bandwidth decreases dramatically, imposing computation and network (e.g. due to storing of frames in a distributed file system) overheads may affect the frame rate (display frequency) even more significantly. Figure 16 shows the comparison between the frame rates without recording and the frame rates with recording (recording every frame) under different available network bandwidths. We observe that the overhead (signified by drop in frame rates) imposed by recording the whole session is very small when the available bandwidth is small. In other words, recording the entire session does not contribute to large performance degradation when network bandwidth becomes a bottleneck. This is counter intuitive to our reasoning presented earlier. However, this can be explained as follows. When the network bandwidth is low, the system generates/transmits fewer frames per second to remote recipients. The low frame rate results in residual computation cycles on the system which are *under utilized*. These computation cycles can be utilized for storing the frames in the database without affecting the frame rate. On the other hand, high available network bandwidth allows the system to generate/transmit larger number of frames to remote recipients. The process for storing the frames now have to compete with the process for generating/displaying/transmitting the video frames, resulting in drop in frame rate to almost half of the original, as shown in figure 16.

Another concern in recording video sessions is the storage bandwidth requirements for recording the video frames. It is possible not to record every frame into the database, and still maintain the quality of video replay. We have conducted experiments to observe the effects of changing recording frequencies on sizes of recorded video sessions (Figure 17). As the recording frequency decreases, there is an exponential decay in size of the stored VC session data. As the available network bandwidth increases, the rate at which the frame can be delivered to the remote sites also increases. Therefore, more frames are stored with the same recording frequency (time interval), thereby increasing the size of the recordings. Here, the recording frequency takes the form of time interval between two consecutive frames to be stored, hence is actually the reciprocal of the time interval.

For a particular network bandwidth, we observe that the size of the stored video session is constant up to a certain value of time interval for recording frames (recording frequency), beyond which it falls exponentially as the time interval between consecutive frames to be stored increases. We call this a turning point t_d for a particular network bandwidth d (see figure 17). For time interval less than t_d, the recording frequency is higher than the display frequency, i.e. the frames are to be stored at a rate greater than those at which they are generated. This will result in duplicates of a frame being stored. However, in our implementation of the recording module, one frame is stored only once but every frame is stored up to t_d time interval. This is achieved by storing the unique timestamp of the encoded frame in the database, and storing a frame only if its timestamp is different from that of the most recently stored frame. To summarize, each frame is stored without duplicates for any time interval between two frames to be stored less than t_d. Beyond this point, the recording frequency is smaller than the display frequency, and hence a subset of the generated frames is stored in the database.

References

[1] Ron Frederick. Experiences with real-time software video compression. In *Proceedings of the Packet Video Workshop*, Portland, Oregon, September 1994.

[2] Didier Gall. MPEG: A Video Compression Standard for Multimedia Applications. *The Communication of the ACM*, 34(10), Oct. 1991.

[3] Winston William Hodge. *Interactive Television: a comprehensive guide for multimedia technologies*. McGraw-Hill, 1995.

[4] Ronald J. Vetter. Videoconferencing on the Internet. *IEEE Computer*, 28(1):77–79, January 1995.

[5] Gregory K. Wallace. The JPEG Still Picture Compression Standard. *The Communication of the ACM*, 34(10), Oct. 1991.

[6] A. V. Aho, J. E. Hopcroft, and J. D. Ullman. *The Design and Analysis of Computer Algorithms*. Adison-Wesley, Reading, MA, 1974.

[7] Nikitas A. Alexandridis. Adaptable software and hardware: Problems and solutions. *IEEE Computer*, 19(2), February 1986.

[8] A. Avizienis. Fault-tolerant systems. *IEEE Transactions on Computers*, C-25(12):1304–1312, December 1976.

[9] Bharat Bhargava and Shirley Browne. Adaptability to failures using dynamic quorum assignments. Technical Report CSD-TR-886, Purdue University, June 1989.

[10] P. A. Bernstein and N. Goodman. Concurrency control in distributed database systems. *Computing Surveys*, 13(2):185–221, 1981.

[11] Daniel Barbara, Hector Garcia-Molina, and Annemarie Spauster. Polices for dynamic vote reassignment. In *Proceedings of the 6th International Conference on Distributed Computing Systems*, May 1986.

[12] Bharat Bhargava. Performance evaluation of reliability control algorithms for distributed database systems. *Journal of Systems and Software*, 3:239–264, July 1984.

[13] Bharat Bhargava. Transaction processing and consistency control of replicated copies during failures. *Journal of Management Information Systems*, 4(2):93–112, October 1987.

[14] Andrew Black, Norman Hutchinson, Eric Jul, Henry Levy, and Larry Carter. Distribution and abstract types in Emerald. *IEEE Transactions on Software Engineering*, SE-12(12), December 1986.

[15] Jon Louis Bentley and Catherine McGeoch. Worst-case analysis of self-organizing sequential search heuristics. In *Proceedings of 20th Allerton Conference on Communication, Control, and Computing*, pages 452–461, University of Illinois, Urbana-Champaign, October 1982.

[16] D. Z. Badal and W. McElyea. A robust adaptive concurrency control for distributed databases. In *Proceedings of IEEE INFOCOM*, pages 382–391, 1984.

[17] Bharat Bhargava, Enrique Mafla, John Riedl, and Bradley Sauder. Implementation and measurements of an efficient communication facility for distributed database systems. In *Proceedings of the 5th IEEE Data Engineering Conference*, Los Angeles, CA, February 1989.

[18] Andrew D. Birrell and Bruce Jay Nelson. Implementing remote procedure calls. *ACM Transactions on Computer Systems*, 2(1):39–59, February 1984.

[19] Bharat Bhargava, Paul Noll, and Donna Sabo. An experimental analysis of replicated copy control during site failure and recovery. In *Proc. of the 4th IEEE Data Engineering Conference*, pages 82–91, Los Angeles, CA, February 1988.

[20] Bharat Bhargava and John Riedl. A model for adaptable systems for transaction processing. In *Proceedings of the 4th IEEE Data Engineering Conference*, pages 40–50, Los Angeles, CA, February 1988.

[21] Bharat Bhargava and John Riedl. The RAID distributed database system. *IEEE Transactions on Software Engineering*, 16(6), June 1989.

[22] Bharat Bhargava, John Riedl, and Detlef Weber. An expert system to control an adaptable distributed database system. Technical Report CSD-TR-693, Purdue University, May 1987.

[23] David. R. Cheriton and W. Zwaenepoel. Distributed process groups in the V kernel. *ACM Transactions on Computer Systems*, 3(2):77–107, May 1985.

[24] Susan B. Davidson, Hector Garcia-Molina, and Dale Skeen. Consistency in partitioned networks. *ACM Computing Surveys*, 17(3), September 1985.

[25] K.P. Eswaran, J.N. Gray, R.A. Lorie, and I.L. Traiger. The notions of consistency and predicate locks in a database system. *Communications of the ACM*, 19(11):624–633, November 1976.

[26] Ahmed K. Elmagarmid and Abdelsalam Helal. Supporting updates in heterogeneous distributed database systems. In *Proc. of the 4th IEEE Data Engineering Conference*, pages 564–569, Los Angeles, CA, February 1988.

[27] Hector Garcia-Molina. Elections in a distributed computing system. *ACM Transactions on Computer Systems*, C-31(1):48–59, January 1982.

[28] Joseph A. Goguen. Reusing and interconnecting software components. *IEEE Computer*, 19(2), February 1986.

[29] Jim Gray. The transaction concept: Virtues and limitations. In *Proc of the VLDB Conference*, Cannes, France, September 1981.

[30] Maurice Herlihy. Dynamic quorum adjustment for partitioned data. *ACM Transactions on Database Systems*, 12(2):170–194, June 1987.

[31] S. Kartashev and S. Kartashev. Guest editor's introduction: Design for adaptability. *IEEE Computer*, pages 9–15, February 1986.

[32] Charles Koelbel, Fady Lamaa, and Bharat Bhargava. Efficient implementation of modularity in RAID. In *Proceedings of the USENIX Workshop on Experiences with Building Distributed (and Multiprocessor) Systems*, Ft. Lauderdale FL, October 1989.

[33] W. H. Kohler. A survey of techiques for synchronization and recovery in decentralized computer systems. *AMC Computing Surveys*, 13(2):149–183, June 1981.

[34] H.T. Kung and J. Robinson. On optimistic methods for concurrency control. *ACM Transactions on Database Systems*, 6(2):213–226, June 1981.

[35] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM*, 21(7):558–564, July 1978.

[36] G. Lausen. Concurrency control in database systems: a step towards the integration of optimistic methods and locking. In *Proceedings of the ACM Annual Conference*, pages 64–68, October 1982.

[37] C. H. Papadimitriou. The serializability of concurrent database updates. *Journal of the ACM*, 26(4):631–653, October 1979.

[38] Michael L. Powell and Barton P. Miller. Process migration in DEMOS/MP. In *Proceedings of the 9th ACM Symposium on Operating Systems Principles*, 1983.

[39] Gerald J. Popek and Bruce J. Walker. *The LOCUS Distributed System Architecture*. The MIT Press, 1985.

[40] B. Randell. System structure for software fault tolerance. *IEEE Transactions on Software Engineering*, SE-1(2):220–232, June 1975.

[41] Richard F. Rashid. Threads of a new system. *Unix Review*, 4(8):37–49, August 1986.

[42] D.P. Reed. Implementing atomic actions on decentralized data. *ACM Transactions on Computer Systems*, 1(1):3–23, February 1983.

[43] D. Skeen. Nonblocking commit protocols. In *Proc of the ACM SIGMOD Conference on Management of Data*, pages 133–147, Orlando, Florida, June 1982.

[44] Amit P. Sheth and Ming T. Liu. Integrating locking and optimistic concurrency control in distributed database systems. In *Proceedings of the 6th International Conference on Distributed Computing Systems*, May 1986.

[45] D. Skeen and M. Stonebraker. A formal model of crash recovery in a distributed system. *IEEE Transactions on Software Engineering*, SE-9(3), May 1983.

[46] P. Wegner. Capital-intensive software technology. *IEEE Sofware*, 1(3):7–45, July 1984.