A Distributed Framework for Collaborative Annotation of Streams

Tao Huang, Shrideep Pallickara, Geoffrey Fox
Department of Computer Science, Community Grids Lab
Indiana University, Bloomington, IN, 47404
{taohuang, spallick, gcf}@indiana.edu

ABSTRACT

Research groups and software companies have developed a number of multimedia collaboration tools such as Access Grid and Vannotea to archive collaborative objects such as audiovisual communications and digital annotations. Most of these tools are designed to process multimedia data streams, and it is not easy for their users to extend or modify them to support other types of data streams such as those generated by earthquake sensors and medical instruments. It is challenging to design and develop a system that supports creating, sharing and replaying annotations on user specified data streams. In this paper, we make a survey of several popular collaboration and annotation tools, and then present our prototype of a distributed framework that supports collaborative annotation on generic data streams. It supports basic annotation operations on the stream data, and it also provides a set of capturing and rendering interfaces that simplify the procedure of adding support for new types of data streams.

KEYWORDS: Collaborative Annotation, Multimedia, Data Stream Interfaces, Distributed Framework

1. INTRODUCTION

Most existing collaboration systems [1-4] can be categorized into two major classes: audiovisual based or digital document based. It becomes difficult when people are required to do collaborative work on new types of real time streaming data. For example, a doctor at Bloomington hospital may want to discuss with his co-workers in Indianapolis about a patient's condition. They may use a videoconferencing tool to communicate verbally or even do collaborative annotations on some X-ray scanning images of the patient. Doctors in Indianapolis nevertheless cannot see the real time heartbeat readings or blood pressure on the monitor of the medical instrument at Bloomington. Though we could solve this problem through some remote display sharing tools, it disables the mutual communication, which causes obstacles to a timely diagnosis. It would be convenient if the collaboration tool they are using can accept those medical readings, transfer them to the remote site over the internet and render them as requested.

This paper describes a novel prototype system developed by the Community Grids Lab at Indiana University Bloomington to solve above problem. It is implemented based on two other projects of the same lab: GlobalMMCS [2][5] and Naradabrokering [6]. We use the media module of the GlobalMMCS project, which is implemented on top of the Sun's JMF [7] library, to enable capturing and rendering of live multimedia streams from web cameras and microphones. Encoded streaming data are transmitted and disseminated through events within the Naradabrokering network. In order to make it simple to support new types of data streams, we analyze generic behaviors of stream processing and define a set of interfaces helping users implement their own capturing sources and rendering players. The system is also designed to have basic fault tolerance on system failures, two recovery strategies are used to deal with local and remote node malfunctions. Details will be explained in later sections.

This paper is organized as follows. Section 2 is a brief survey of popular existing annotation systems. Based on the analysis of these systems, we summarize our objectives of the collaboration framework in section 3. In section 4, we describe the architecture and important components of the prototype system. Annotation management is explained in details in section 5. After analyzing results of some preliminary experiments in section 6, we conclude and present our future plans.

2. RELATED WORK

Distributed collaboration and annotation systems [8-12] have been developed in the past decade all around the world. These systems have been designed to service different aspects of collaboration. H.323 systems such as

Polycom and Tandberg dominate the videoconferencing market and they do provide reliable audiovisual communications in the heterogeneous network. As a free alternative, Access Grid [1] is very popular in the academic community. Scientific discussions and lectures are being held on this platform almost every day. Besides videoconferencing, document sharing and annotation is another major requirement of current collaborative annotation systems. Tools such as Google Docs [3] and Microsoft Office Live Workspace [4] are invented to facilitate online document based work. Recently all these tools tend to share their features. For instance, Access Grid has some basic document sharing capabilities via its web portal while Good Doc users can video chat with each other through the new Gmail feature.

In this section, we make a brief survey of popular distributed annotation and collaboration tools. By analyzing them, we try to find out important features that can be introduced to our prototype.

Microsoft research released its annotation system MRAS [11] in 2000, the system was designed to help Microsoft employees gain better training experience through asking questions on pre-recorded lecture videos. The questions are anchored on the multimedia content and answered by the instructors asynchronously. Since the questions can be synchronously replayed with the class content, students that have similar questions at the same time spot will benefit a lot from reading answers to the previous question. Collaboration is achieved through discussions on the questions and their answers. MRAS doesn't support live video feeds and students who are watching the same video streams could not exchange their thoughts in the real time.

IBM's Mpeg-7 annotation tool – VideoAnnEx [12] was also released in 2000. It can parse Mpeg video files and segment them into small shot units. Each shot unit can be annotated with a description from three default categories: static scene, key object and event. All shot units are stored into a XML file as well as their descriptions/annotations following the Mpeg-7 standard. Users can search among the descriptions and replay the video shots alongside the description they are looking for. VideoAnnEx is a stand-alone annotation program that cannot accept live video feeds either, and it does not support sharing and manipulating video streams among distributed users. It can merely process Mpeg-1 and Mpeg-2 video files and the descriptions are limited to three pre-defined categories. It is difficult to extend the system without modifying the underlying source.

A group of researchers from University of Queensland invented Vannotea [13] to help facilitate collaborative video indexing, annotation and discussion of video contents in the distributed broadband environment. It supports most features that VideoAnnEx has and provides more flexibility on the metadata of video segments. Vannotea users are able to save, browse, retrieve and share both objective descriptions of the video files as well as subjective annotations on them. The videos files are still limited to Mpeg-2 format and users can only create text descriptions.

eSports [14] developed by Community Grids Lab is another attempt to enable collaborative annotation on multimedia content over the distributed network, especially the grid-computing network. It enriched the annotation on multimedia contents from simple text to more diverse forms such as graphic shapes, audio/video clips. As its name indicates, eSports system aims to help sport coaches train their trainees remotely through vocal and graphic annotations on real time or archived video streams. Coaches can take snapshots of sample gestures in the video and comment on them to help students understand their classes. Annotations and video streams are archived using Naradabrokering storage service and can be replayed synchronously based on their timestamp property. Since the streams are stored as a series of Naradabrokering events rather than large video files, users can ask to replay any part of the stream without loading all related events. Live chat is also implemented to improve the real time communication in the system.

All systems described here provide video annotation capabilities and support synchronous replaying of annotations/descriptions alongside the video content. MRAS and VideoAnnEx are stand-alone programs that enable asynchronous communication and searching in annotation, while Vannotea and eSports spent more efforts in supporting annotation on real time video streams in distributed environments. None of them has considered the ability of supporting non-multimedia streams, and it is difficult to add this new feature to them without modifying their sources codes.

3. OBJECTIVES

From the survey and analysis in the previous section, we can determine basic objectives of our collaboration framework for stream annotation. It should be able to support creating, archiving and replaying multiple forms of annotations on either real time or prerecorded data streams without knowing their characteristics. The system should support both synchronous and asynchronous communications on both annotations and content streams. As a distributed system, a robust session management is required to make the system tolerant to possible hardware or network failures. Capabilities of recovering from disastrous situations are also required. In addition the system needs to support following features that help expand its application fields:

- Support processing multimedia streams in different formats/codecs.
- A generic data stream processing API, which can help users extend the system with their own stream capturing sources and rendering players.
- Support annotating, commenting and discussion on live data streams. Users in the same session should be able to watch each other's annotation in the real time instead of loading them from the archiving repository.
- A simple interface that helps in saving, searching and sharing annotation among distributed users easily.
- The system should support various types of clients from handheld devices to streaming clients.

4. ARCHITECTURE

Figure 1 below depicts a typical scenario of using our prototype. A stream annotator is feeding a live video stream to the system and making notes on it. Client A and B are live collaborators in the same session and they are able to ask questions on the video stream while it is being played. Another client using a handheld device is watching the collaboration activities between the annotator and client A and B. Session information, annotations and stream data are transmitted and exchanged using Naradabrokering events. All events are automatically stored into the stream repository for later replays. Different metadata are stored in each event's header, and information within them facilitates functions such as stream synchronization and system recovery.

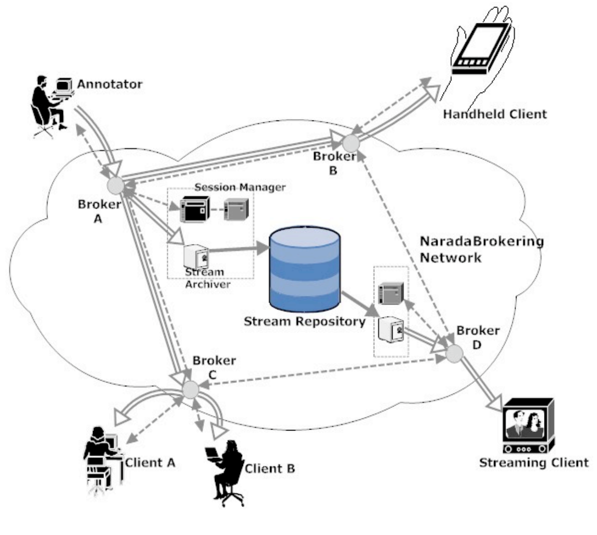


Figure 1. System Architecture

In the above picture, we can find three major components of the system: Session Manager, Annotation Client and Stream Archiver. Session Manager maintains all session related information such as client joining or leaving. The client is responsible for generating content streams as well as receiving and replaying streams from other clients. It also parses annotation events to reproduce actual annotations on the content stream. Stream Archiver is spawned by Session Manager to archive live streams in the stream repository, either locally or remotely. It is also responsible for retrieving archived streams as per the client's requests.

4.1. Session Management

Due to the pub/sub nature of the Naradabrokering platform, we use heartbeats to manage the session information in the system. Each component in the system continuously publishes its own heartbeat event to public channels. All clients will monitor heartbeat events in the session channel and maintain their own copies of the session status, i.e. list of active clients in current session. Unresponsive clients will be removed from the list if other clients cannot hear from them for more than three seconds. Session Manager monitors the session channel as well and periodically broadcasts its own client list as the standard for participating clients to synchronize their lists with. Session Manager will also monitor the service channel to control active stream archivers and remove unnecessary ones. A status report will be generated and stored in the local file system and remote stream repository after a customizable period of time.

As the core management component of a distributed system, Session Manager should be available all the time and be able to recover from disastrous situations such as program crashes and power outages. We use two strategies to maintain such durability: Local recovery and Remote recovery.

Local recovery: Alongside the running Session Manager, a daemon process (gray manager in Figure 1) keeps collecting session information as other clients do. It starts taking over the management responsibility when the running manager freezes and stops publishing standard heartbeat. It will kill the original manager process, changes its own status by parsing the latest status report on the file system and create another daemon process to take over its previous job. Since clients will not check the source of the standard heartbeat, they will not know the manager has been replaced.

Remote recovery: We could not apply local recovery if there were hardware problems or power outages on the running manager machine. In such circumstances, all clients will find a best machine among them by exchanging and comparing their hardware information. The most appropriate client will create the manager process, adjust its status according to the remote status report and start collecting information from both the session and service channels.

4.2. Annotation Client

Figure 2 below shows three layers of our annotation client: Transmission layer, Logic layer and Presentation Layer from the bottom up. Each takes its own responsibility of processing the streaming data.

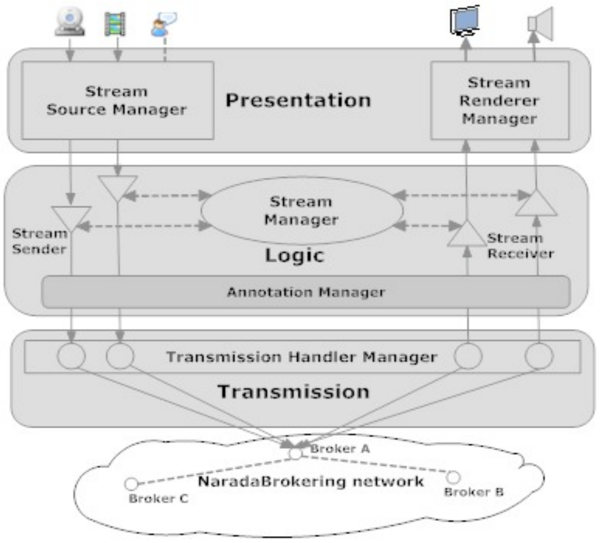


Figure 2. Three Layers of the Annotation Client

The Transmission layer is responsible for creating and managing actual data transmission handlers (called DataTransmitter in the source). Each transmission handler contains a pair of Naradabrokering event consumer and publisher, and it subscribes itself to a particular topic specified by the ID of the stream it operates on. In order to minimize the cost of handler creation and termination, a pool of handlers (around 5 handlers) are created during the start up of the client. Similar to the Java thread pool, transmitting handlers are assigned and recollected by a handler manager.

The Logic layer works as an important mediating layer between the Transmission layer and the Presentation Layer. For stream capturing and rendering, a stream sender or receiver will be created to connect a stream source/renderer from the presentation layer with a transmitting handler from the transmission layer and start the processing. There is a stream manager in this layer to manage all active senders and receivers. The Annotation manager also sits within this layer to associate and synchronize content data streams with the annotation streams.

The Presentation layer is the upper-most layer and it contains the graphic user interface, stream source and renderer managers. Similar to the DataSource class in the JMF library, a stream source is an object that can generate real time data constantly when it is started. It can be paused or stopped. Stream renderers are used to decode received stream data and display the content on the screen.

Figure 3 below is the class diagram that shows the interrelationships between the stream source/renderer interfaces and the stream sender/receiver classes.

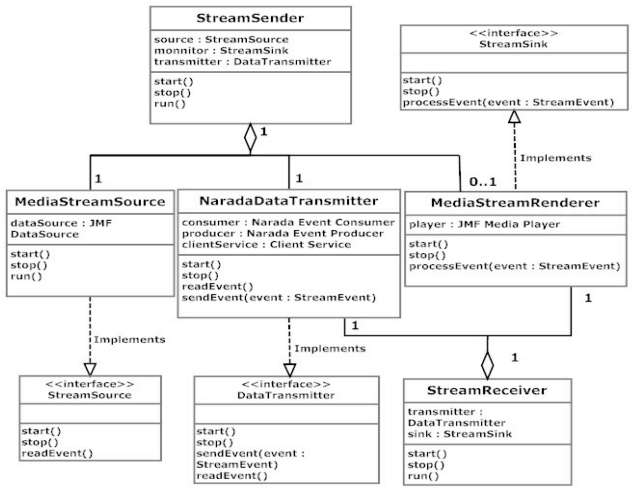

StreamSender
source : StreamSource
monnitor : StreamSink
transmitter : DataTransmitter
start()
stop()
run()
<<interface>> StreamSink
start()
stop()
processEvent(event : StreamEvent)
1
Implements
1
1
0..1
MediaStreamSource
dataSource : JMF DataSource
start()
stop()
run()
NaradaDataTransmitter
consumer : Narada Event Consumer
producer : Narada Event Producer
clientService : Client Service
start()
stop()
readEvent()
sendEvent(event : StreamEvent)
MediaStreamRenderer
player : JMF Media Player
start()
stop()
processEvent(event : StreamEvent)
1
1
Implements
Implements
1
<<interface>> StreamSource
start()
stop()
readEvent()
<<interface>> DataTransmitter
start()
stop()
sendEvent(event : StreamEvent)
readEvent()
StreamReceiver
transmitter : DataTransmitter
sink : StreamSink
start()
stop()
run()

Figure 3. Class Diagram of Stream Processing Interfaces

Since the stream source/sink interfaces in above picture only define the generic behaviors of a real time data stream, users can easily write their own stream sources and renderers to extend the system. They just need to implement those interface methods in their existing source/rendering classes and compile them with the client source. This will save a lot of effort as opposed to understanding and modifying source codes of the entire system. In our current release, we have implemented several stream sources such as video/audio capturing source, file capturing source and screen capturing source and their corresponding renderers. With the help of the GlobalMMCS media module, our system supports various video/audio formats on different operation systems. We list them in the table 1 below.

Table 1. Supported Multimedia Formats

OS	Video	Audio	Screen Capture
Windows	H.261, H.263, DIVX, JPEG	ULAW, GSM, DVI, G729	H.261, DIVX, JPEG

Linux	H.261, H.263, JPEG	ULAW, GSM, DVI	N/A
Mac	H.261, JPEG	ULAW, GSM, DVI	N/A

4.3. User Interface

Figure 4 is a snapshot of our annotation client running on Windows XP. We implement the client using SWT library [15], an OS-independent widget toolkit from the Eclipse project. The client comprises a tree based client list and three composite panels. Each panel can be maximized to show as much information as possible.

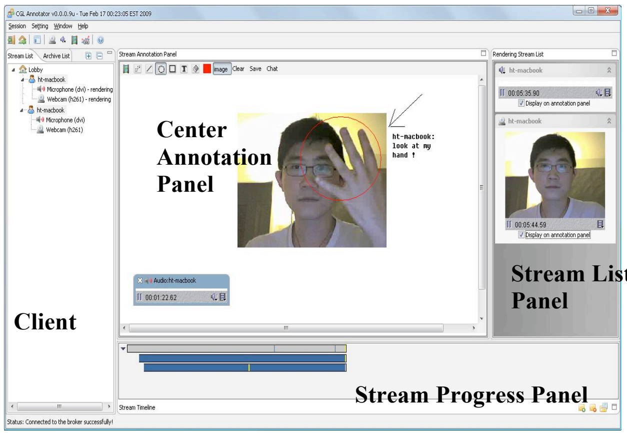


Figure 4. A Snapshot of The User Interface

The client list on the left displays all participating clients in the same session. The user can open any data stream (video steam in the snapshot) being sent by a client. Once the receiver of this data stream is created and started successfully, the renderer window will be displayed in the stream renderer list on the right panel. Users can also select to create a clone of the playing renderer to the center panel by checking the checkbox underneath it. A stream progress widget is also created on the progress panel below once the clone starts playing. Unlike the original renderer window on the right, the cloned renderer can be positioned anywhere on the center panel and the user is able to either rewind or fast forward the playing content by dragging the progress indicator on its stream progress widget.

Alongside the client list, there is an archive list that only displays information of data streams stored by stream archivers. Users can apply all available operations on these archived streams as if they were normal live streams. There is no difference between them and the live stream since they are just duplicates of the stored live streams from the event repository, loaded and published by stream archivers. More details of archiving and replaying streams will be explained in the next section.

There are two modes of rendering received data streams in our client: live and buffered. The first mode is the default one. Events of an incoming data stream are temporarily stored in a small in-memory buffer to reduce the influence of possible event losses in the transmission. Sometimes, it would be useful if users could rewind the playing content to the exact position that they want to insert annotations at. This requires enabling the buffered mode of rendering the stream. As depicted in the following figure 5, decoded video frames are written into a temporary file and can be retrieved from any time spot based on the frame rate information inside the stream's video codec. When the user makes a rewind operation on the current stream progress, a buffered stream source is created at the correct playing time and started to read the correct video frames from the buffer file for the stream renderer to display. A reading clock controls the speed of the buffered source and makes sure that it generates frames at the right frame rate. Despite the disk access overhead introduced here, this feature enables annotation on live video streams while they are being watched.

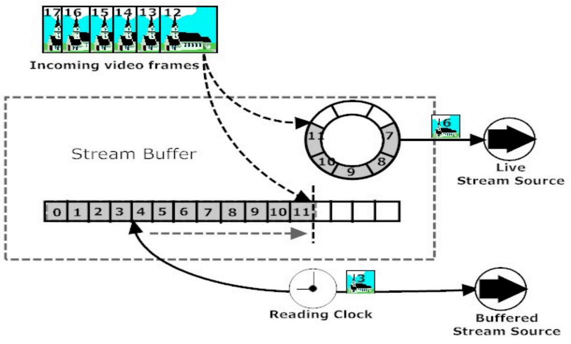


Figure 5. A Running Example of the Stream Buffer

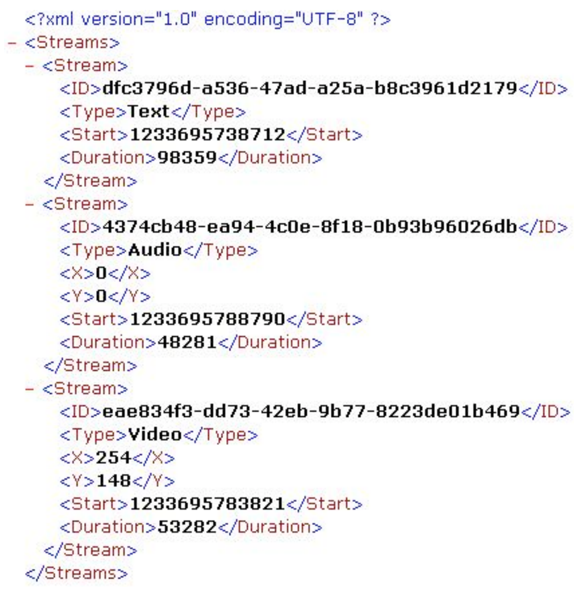
4.4. Stream Archiver

Stream Archiver is one of the most important components in the system. It takes the responsibilities of archiving live data streams and replaying them per the client's requests. In our current implementation, the archiver stores every stream event into a remote database alongside the meta-information such as time stamp and stream description in the event's header. When a request of replaying a particular data stream is received, the corresponding archiver will read all stream events based on time range information within the request. Events will be published to a specific replaying topic based on the request ID known by the requesting client.

As explained in the previous section, Stream Archiver is monitored and controlled by the Stream Manager. When a sending stream is stopped, Stream Manager will terminate its corresponding archiver unless there are some clients requesting to replay this stream.

5. ANNOTATION MANAGEMENT

In Figure 4, you can see that there is a stream progress panel on the bottom of the client. It allows users to control the rendering of data streams on the center annotation panel and create annotations on them. The stream progress widget displays the length and playing progress of the stream. When an annotation is created, information of all the stream renderers on the annotation panel is stored into a XML DOM object and each renderer starts to update this object with its newest progress. Following is an XML example generated from a simple annotation DOM object.



```
  <?xml version="1.0" encoding="UTF-8" ?>
- <Streams>
  - <Stream>
      <ID>dfc3796d-a536-47ad-a25a-b8c3961d2179</ID>
      <Type>Text</Type>
      <Start>1233695738712</Start>
      <Duration>98359</Duration>
    </Stream>
  - <Stream>
      <ID>4374cb48-ea94-4c0e-8f18-0b93b96026db</ID>
      <Type>Audio</Type>
      <X>0</X>
      <Y>0</Y>
      <Start>1233695788790</Start>
      <Duration>48281</Duration>
    </Stream>
  - <Stream>
      <ID>eae834f3-dd73-42eb-9b77-8223de01b469</ID>
      <Type>Video</Type>
      <X>254</X>
      <Y>148</Y>
      <Start>1233695783821</Start>
      <Duration>53282</Duration>
    </Stream>
  </Streams>
```

Figure 6. Annotation Dom Object in plain XML

As seen in the above picture, there are no actual stream events stored in this XML file. We only record information that represents the layout of all active streams in the annotation panel, for example, position of the renderer on the center annotation panel, absolute start time of the stream and its duration. All this information will be used to reconstruct the annotation scenario later on.

When the annotation owner closes the annotation, an XML copy of the annotation object will be saved remotely in the annotation storage. A local copy is also created as backup for fast accessing. When the user decides to replay the annotation he creates, the client will first check the local file system before asking the remote repository. The Dom object will be parsed and created from the XML file and all renderers will be regenerated as well as their annotation.

6. PRELIMINARY EXPERIMENTS

Being at the early stage of the prototype development, we are more interested in making the system stable and capable of dealing with large number of data streams at the same time. Therefore we did some preliminary stress tests on the stream archiver by feeding a different number of multimedia streams in different formats at the same time. CPU usages of the running archiver process are logged and displayed in the following Figure 7.

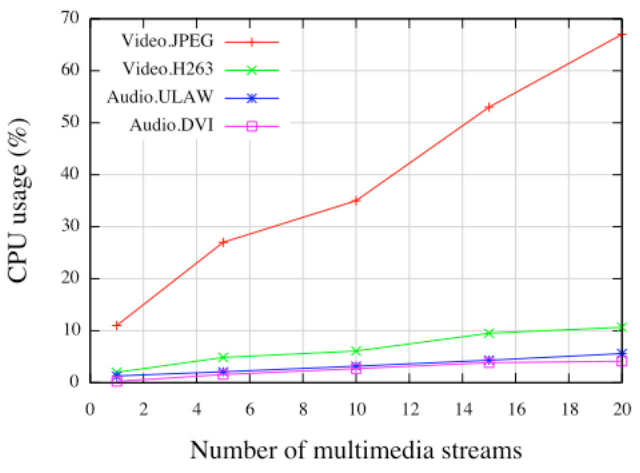


Figure 7. CPU Usages of A Stream Archiver Archiving Different Multimedia Streams

The experiments were done on an Intel Pentium 4 machine with a 3.40GHz CPU and system memory of 1.75G. The results show us that the stream archiver works pretty well on streams that are made up of events with small payloads, such as audio streams and highly compressed video stream in the figure. Less than 10% CPU was used to process 20 simultaneous Video.H.263 streams. Since a large event payload requires more copy instructions and system I/Os, it is not hard to explain why CPU usages were so high when the stream archiver tried to archive those Video.JPEG streams. We also notice that the CPU usages of brokers in the Naradabrokering system were also at a quite high level when they are transmitting Video.JPEG streams.

Our system has a built-in whiteboard (see Figure 4) to support free-hand drawing annotation as eSports does. It is important that drawings such as lines, shapes and inserted images are displayed timely on remote clients, especially when users are working on real time data streams. Delayed or disordered annotations will cause problems to the real time communication. We tested our system by sending large amounts of free-hand whiteboard events in one second while system users are playing different types of multimedia streams. We record the time difference between each event's creation time and rendering time at remote clients. The Average of all differences recorded in the same test is used as the final result.

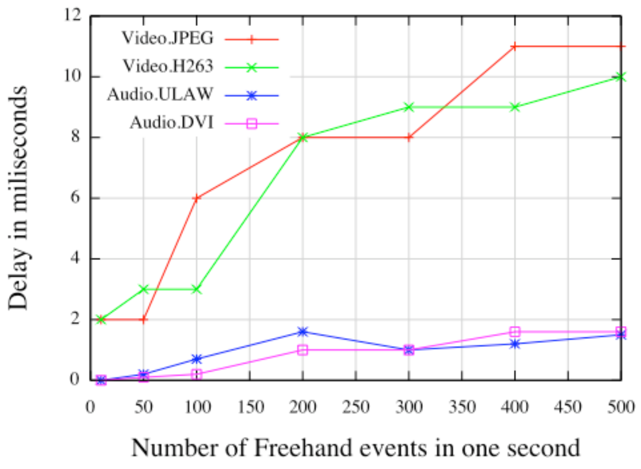


Figure 8. Time Delays of Freehand Whiteboard Events

Though ascending, time delays caused by the system are still much lower than the required perception level of delay (200-400ms for video streams) in a cooperation system [16]. Distributed users will not have any problems on whiteboard annotations in the system while they are cooperating on supported real time data streams.

7. SUMMARY AND FUTURE WORKS

In this paper, we introduce a framework system that supports collaborative annotation on generic data streams. It supports sending, browsing, rendering and annotation on real time data streams in distributed environments and our experiment results show that it works properly for compressed data streams under high stress circumstances.

This system expands its scope of application through generalizing the procedure of data stream processing and defining basic stream capturing and rendering interfaces. Users are able to quickly extend the system by writing their own stream sources/renders. Through implementing those interface methods, we can support more types of data streams other than mere multimedia ones in the system, which makes it more capable of satisfying diverse application requirements. The system also provides a simple user interface to ease the manipulation of streaming data and it also supports annotation on live data streams via local stream buffers.

Our next step is to continue the development of this prototype to improve its stability. More stream sources and renders will be added to the system to support data streams generated by non-multimedia sources such as earthquake sensors, handheld devices and medical instruments. A configuration detector will be added to the system to simplify the recognition of new "StreamSource" and "StreamSink". We plan to standardize our annotation metadata format into Mpeg-7 compatible version so that we can have more accurate search functionality. After this, a Web 2.0 styled portal and web client (based on Ajax or Adobe Flex) may also be added to the system to facilitate customized annotation search and viewing on different computing platforms.

REFERENCES

[1] Access Grid project. Available from http://www.accessgrid.org

[2] GlobalMMCS project. Available from http://www.globalmmcs.org

[3] Google Doc. Available from http://doc.google.com

[4] Microsoft Office Live Workspace. Available from http:// http://workspace.officelive.com

[5] Wenjun Wu, Tao Huang, Geoffrey Fox, "Building Scalable and High Efficient Java Multimedia Collaboration," *Proceedings of IEEE 2006 International Symposium on Collaborative Technologies and Systems* CTS 2006 Las Vegas May 14-17 2006

[6] S. Pallickara and G. Fox, "NaradaBrokering: A Middleware Framework and Architecture for Enabling Durable Peer-to-Peer Grids," *Proceedings of ACM/IFIP/USENIX International Middleware Conference* Middleware-2003. pp 41-61.

[7] Sun Java Media Framework API. Available from http://java.sun.com/javase/technologies/desktop/media/jmf

[8] Carrer, M., Ligresti, L., and Little, T. D, "A Tcl/Tk-based video annotation engine," *Proceedings of the 5th Conference on Annual Tcl/Tk Workshop 1997 - Volume 5*, USENIX Association, Berkeley, CA, 30-30.

[9] A. Savakis, P. Sniatala, and R. Rudnicki, "Real-time Annotation using MPEG-7 Motion Activity Descriptors," *MIXDES 2003*, Lodz, Poland, June 2003.

[10] Wei Ren and Sameer Singh, "An Automated Video Annotation System," *Pattern Recognition and Image Analysis*, ICAPR 2005, LNCS 3687, pp. 693 – 700, 2005.

[11] Bargeron, D., Gupta, A., Grudin, J., Sanocki, E., Li,

F, “Asynchronous Collaboration Around Multimedia and its Application to On-Demand Training,” *Proceedings of the 34th Hawaii International Conference on System Sciences (HICSS-34)*, January 3-6, 2001, Maui, Hawaii

[12] J. R. Smith and B. Lugeon, "A Visual Annotation Tool for Multimedia Content Description," *Proc. SPIE Photonics East, Internet Multimedia Management Systems*, November 2000.

[13] R. Schroeter, J. Hunter, and D. Kosovic. “Vannotea - A Collaborative Video Indexing, Annotation and Discussion System for Broadband Networks,” *Knowledge Markup and Semantic Annotation Workshop, K-CAP 2003*. Sanibel, Florida. October 2003.

[14] Gang Zhai, Geoffrey Fox, Marlon Pierce, Wenjun Wu, Hasan Bulut, “eSports: Collaborative and Synchronous Video Annotation System in Grid Computing Environment,” *Proceedings of IEEE International Symposium on Multimedia (ISM2005)*, Pages 95-103, IEEE Computer Society, December 12-14, 2005 Irvine, California, USA

[15] SWT: The Standard Widget Toolkit. Available from http://www.eclipse.org/swt/

[16] Leping Huang, Mitsuharu Iijima, Kaoru Sezaki, “A Survey on Human Perception of Delay in a Cooperation System”, *IEICE Communications Society Conference 1999*, B-11-12, Chiba, Japan, Sep.1999