Service-Oriented Architecture for Building a Scalable Videoconferencing System

Ahmet Uyar, Wenjun Wu, Geoffrey Fox

ABSTRACT

In this paper, we propose a service-oriented architecture to develop a scalable videoconferencing system, GlobalMMCS, based on a publish/subscribe event brokering network. There are two key ideas behind our approach for building a scalable videoconferencing system. First one is to identify and separate tasks performed in a videoconferencing system, and design independent scalable components for each task. Second one is to coordinate system components by providing service based interactions.

We identified that there are three main tasks in a videoconferencing system: media (audio/video) distribution, media processing and meeting management. We use a publish/subscribe event brokering system, NaradaBrokering, to deliver all media and data messages exchanged in the system. It provides a unified scalable solution for content distribution. We developed a scalable media processing framework that provides media processing services such as audio mixing, video mixing, and image grabbing. In addition, we have designed meeting managers to start/stop meetings and control the resources necessary for them.

All service providers announce the service descriptions upon a request. Service schedulers use these descriptions to determine the best available service providers based on the policies set by users. Consumers exchange request/response messages with producers when executing the services. In addition, our system provides a convenient framework for third party developers to add new services. We have implemented a prototype of this system and have used it for experimental purposes.

1 INTRODUCTION

The availability of increasing network bandwidth and the computing power provides new opportunities for distant communications and collaborations over the Internet. On one hand, broadband internet connections are spreading rapidly. Even cell phones will have broadband internet access in the near future with the implementations of 3G standards. On the other hand, the usage of webcams and video camera enabled PDAs and cell phones are increasing by many millions every year. Therefore, it is not entirely inconceivable to imagine that the trend in the increasing usage of videoconferencing systems will continue by accelerating. This will require universally accessible and scalable videoconferencing systems that can deliver thousands or tens of thousands of concurrent audio and video streams. In addition to audio and video delivery, such systems should also provide scalable media processing services such as transcoding, audio mixing, video merging, etc. to support increasingly diverse set of clients.

However, developing videoconferencing systems over Internet is a challenging task, since audio and video communications require high bandwidth and low latency. In addition, the processing of audio and video streams is computing intensive. Therefore, it is particularly difficult to develop scalable systems that support high number of concurrent users. Current videoconferencing systems such as IP-Multicast and H.323 can not fully address the problem of scalability and universal accessibility. These systems focused on delivering the best performance and lacked flexible service oriented architecture. That led to the development of low level standards and hardware based systems that are either difficult to be supported universally or inflexible to add resources. We believe that with the advancements in computing power and network bandwidth, more flexible and service oriented systems should be developed to manage audio and video conferencing systems. In this paper, by utilizing the researches in the areas of publish/subscribe messaging systems and service oriented computing, we propose a service-oriented architecture to develop a scalable videoconferencing system, GlobalMMCS, based on a publish/subscribe event brokering network, NaradaBrokering.

The first step when building a scalable videoconferencing system is to analyze and identify the tasks performed in videoconferencing sessions. Then, independently scalable components for each task can be designed. It is also important to coordinate the interactions among these components in an efficient and flexible manner. We identified that there are three main tasks performed in videoconferencing sessions: audio/video distribution, media processing and meeting management. We propose using a publish/subscribe event brokering system as the audio and video distribution middleware. We also propose a service oriented meeting management architecture to organize media processing service providers in a flexible and scalable manner.

When developing such a system, in addition to scalability, there are also some other factors to observe. This system should be fault tolerant to provide continuing services. It should also be easy to dynamically add or remove components. Both small and large size institutions should be able to configure this system according to their needs. Furthermore, this system should be flexible enough for other developers to add new services.

The content of this paper is organized as follows. First, we cover the most widely used current videoconferencing systems. Then, we analyze the videoconferencing sessions and determine the guidelines to develop videoconferencing systems. In section 4, we cover various types of videoconferencing practices. In section 5, we provide the rationale for using a publish/subscribe system as the middleware and give an overview of the event brokering system we use. In section 6, we outline our approach and provide the details.

2 RELATED WORK

Currently, there are videoconferencing systems based on two main standards: IP-Multicast [mult-evol] and H.323 [h323]. SIP [sip] is another standard which is used to establish real-time sessions. It can also be used to implement videoconferencing systems, but it does not propose any architecture for building video conferencing systems.

IP-Multicast is a set of transport level protocols which provide group communications over the Internet. It provides services such as group formations and management, package delivery mechanisms, inter-domain interactions, etc. All these protocols are implemented on routers. Multicast has two very important advantages. First one is its minimal usage of bandwidth. A sender sends one copy of a stream and it is duplicated along the way from sources to destinations when necessary. It avoids sending multiple copies of the same stream on the same link. Therefore, it saves bandwidth. Another advantage of multicast is its ease-of-use. A group of users need to know only the group address to start a meeting. This simplifies the management of meetings significantly. On the other hand, multicast tries to provide a group communication infrastructure for all Internet users. That results in the scalability and manageability problems [mult-evol]. In addition, it lacks widespread support of internet routers and its traffic is blocked by almost all firewalls. Therefore, it is not suitable for systems that serve all internet users. But its group communication approach is very relevant and helpful.

H.323 [h323] is a videoconferencing recommendation from International Telecommunications Union (ITU) for package based multimedia communications systems. It defines a complete videoconferencing system including audio and video transmission, data collaboration and session management. It is heavily influenced by telephony industry and its primary focus is the performance. It provides a binary protocol and many h.323 based systems are hardware based. It seems that scalability was not primarily important for them. Media processing and media distribution are not separated. In addition, they recommend MCU cascading for large scale conferences, which is a very limited approach to get scalability.

3 TASK ANALYSIS IN VIDEOCONFERENCING SYSTEMS

There are three main tasks performed in videoconferencing sessions on server side.

1. **Audio/video distribution:** This includes transferring audio and video streams from source clients to destinations in real-time. This is a challenging task, since those streams require high bandwidth and low latency. It is essential to provide an efficient media distribution mechanism that will route media streams through best possible routes from sources to destinations. Otherwise, unnecessary network traffic might be generated and additional transit delays might be added. This might cause gaps in the communication and result in poor quality of service. When we also add the requirement for scalability, it becomes clear the importance of determining the right distribution mechanism.
2. **Media Processing:** Media processing is another very important task performed in videoconferencing sessions on server side. Some of the more common ones are audio mixing, video merging, media transcoding, stream monitoring, etc. These tasks usually require high computing resources and real-time output. Therefore, they can limit the scalability of such systems severely when implemented poorly. More importantly, they can affect the quality of audio and video distribution if they share the same computing resources with media distribution units. This requires us to separate media processing units from media distribution units completely to be able to provide scalability.
3. **Session management:** Session management includes starting/stopping/modifying videoconferencing sessions. It also includes determining and assigning system resources for these sessions. For example, it includes finding out the right audio mixing unit to be used by a meeting. In addition, it includes for participants discovering/joining/leaving sessions. Contrary to the media distribution and media processing tasks, session management requires little bandwidth and computing resources. However, it is very important to coordinate and distribute the tasks in such sessions. Therefore, it is crucial to design a flexible and scalable session management mechanism.

3.1 Component Design Guidelines

Here we determine the guidelines to be observed when designing system components for the tasks identified above in videoconferencing sessions.

3.1.2 Audio/video distribution

An efficient audio/video distribution system should have a number of characteristics. First of all, the best possible routes must be chosen from sources to destinations when delivering the content. This is important both not to load extra traffic on the network and not to add extra transit delays to packages.

Secondly, audio and video streams need to be replicated only when it is needed along the path from sources to destinations. This saves significant bandwidth. Usually there are multiple participants in a videoconferencing session, when the sender of a stream sends one copy and distribution network replicates it when necessary, that user is removed from the burden of sending many copies. In addition, in the case of a distributed media delivery network, significant bandwidth can also be saved inside this network by replicating the streams at appropriate points.

Thirdly, since audio and video streams are composed of many small sized packages, it is important to add minimum headers to each of these packages. Otherwise, there can be substantial increase in the amount of data transferred. All audio and video transport protocols and codecs are designed to minimize this bandwidth.

Lastly, unreliable transport means should be used whenever possible. Contrary to many data applications, audio and video transfer can tolerate some package loss. Though, it should be kept in minimum to limit the negative affect of the losses on the quality of the communication. These protocols usually provide lower transit delays, since they do not have the extra cost associated with error correction and package retransmission.

3.1.3 Media Processing

Since media processing units require high computing resources, it is essential to provide a scalable framework. When more computing power is needed, new resources should be added easily. In addition, computing resources should be allocated efficiently to avoid starvation of some processing units. Moreover, the media processing framework should be flexible for new services to be added. Since there can be many types of media processing performed in a video conferencing system, one can not implement all. Therefore, it is important to provide a framework for others to add their own services.

3.1.4 Session Management

Since this system should be scalable, messaging semantics among system components should be efficient. In addition, since we want this system to be easy-to-understand, the interactions among system components must be identified clearly. Moreover, addition/removal of components should be transparent to other components in the system.

4 ONLINE MULTIPARTY CONFERENCING

Similar to real life meetings, there can be many types of online meetings. In the simplest form, there can be one speaker and listeners. Only the speaker sends audio and video streams. Listeners are passive and receive the audio and video of the speaker. When they need to interact with the speaker, they utilize some other means such as chat. Therefore, this type of meetings does not require any media processing on server side. They can be implemented easily and scale well. When a publish/subscribe event brokering system is used, listeners only need to know the topics on which speaker is publishing the audio and video streams. They can receive those streams by subscribing to those topics.

Another type of online meetings can be similar to real life open discussion sessions. There can be many speakers in a meeting and anybody can send audio or video streams at anytime. The order of the meeting can be maintained by observing the general rules of courtesy. This is very similar to multicast meetings. Two topics are determined for a meeting. One is for the exchange of audio streams and the other is for the exchange of video streams. This type of meetings also does not require any media processing at server side. There are two disadvantages of this type of meetings. One is its scalability. Since there can be many streams in a meeting, it would be difficult to support large number of participants. It is also not practical to have large number of users in an uncontrolled meeting. There can be many disturbances such as noise and unwanted interruptions. Another disadvantage is its high capacity requirement for clients. Each participant should be able to receive/send multiple concurrent audio and video streams. This prevents many low end clients from joining such meetings.

The third type of meetings can be more structured. Some participants can be dedicated as speakers and some others as listeners. Speakers can have the right to send audio and video streams. But listeners can only receive the audio and video streams sent by speakers without being able to contribute audio or video. However, some listeners can be promoted to speaker role temporarily when they want to ask a question or make a comment. In addition, this kind of meetings can have media processing units to support heterogeneous clients with various capabilities. When implementing this type of meetings, each audio and video stream should be published to a different topic. As a result of

this, users will be able to choose the streams they want. They will not be delivered extra streams that they have not chosen, since that might overwhelm their capacity.

Publish/Subscribe systems provide a convenient medium to implement any of these meeting types. Since the last type of meetings can be considered a more general form of the previous two types, here we focus on the implementation of that kind of meetings. First, we will introduce the rationale to use publish/subscribe systems as the media delivery middleware and briefly cover the characteristics of the publish/subscribe system that we use.

5 MEDIA AND CONTENT DISTRIBUTION NETWORK

Traditionally, multicast is used to deliver audio and video streams for videoconferencing sessions. But, the lack of widespread use and the problems with firewalls/NATs discouraged many people from using it. Many researchers worked on software level multicasting with limited success [end-system, tapestry, bayeux]. On the other hand, we take a novel approach and utilize the researches in the areas of publish/subscribe systems to provide a scalable solution to media distribution problem. We use NaradaBrokering distributed event brokering system for delivery of audio and video streams.

Publish/subscribe systems provide a messaging middleware that decouples producers and consumers on time, space and synchronization [ps-faces, ps-evol]. The decoupling of producers and consumers on time means that consumers do not have to consume messages as they are produced. The middleware can deliver the recorded messages in a later time. For videoconferencing applications this feature can be exploited when archiving and replaying the archives. The decoupling on space means that producers do not have to know anything about the consumers and consumers do not have to know anything about the producers. This opens up the door to implement scalable videoconferencing sessions. Producers publish only one copy of a stream and the brokering middleware delivers them to all subscribers. The decoupling on synchronization means that producers and consumers do not have to be blocked when receiving or sending messages. This lets a client to send and receive many streams at a time.

Since publish/subscribe messaging systems provide reliable group communication services, in addition to audio and video delivery, they can also be used to deliver the control messages exchanged among the distributed components in the system. Moreover, the same messaging infrastructure can be used for collaboration applications [garnet] that increase the quality of user experience considerably in videoconferencing sessions such as chat, file sharing, application sharing, display sharing, etc. Therefore, one messaging system provides a unified content delivery mechanism that simplifies the design and management of our system significantly.

On the other hand, since publish-subscribe systems are not designed to serve real-time multimedia traffic. They are usually used to deliver guaranteed messages by employing reliable transport protocols. In addition, they do not focus on delivering high bandwidth traffic or reducing the sizes of the messages they transfer. It is more important for them to provide more services than saving bandwidth. Each message tends to have many headers related to the content description, reliable delivery, priority, ordering, distribution traces, etc. Many of these services are not important for audio and video delivery. Therefore some additions need to be made to deliver multimedia traffic.

5.1 NaradaBrokering

NaradaBrokering [nb1, nb2] is a distributed publish/subscribe messaging system that provides a scalable architecture and an efficient routing mechanism. It organizes brokers in a hierarchical cluster-based architecture. The smallest unit of the messaging infrastructure is the broker. Each broker is responsible for routing messages to their next stops and also handling subscriptions. In this architecture, a broker is part of a base cluster that is part of a super-cluster, which in turn part of a super-super-cluster and so on. Clusters comprise strongly connected brokers with multiple links to brokers in other clusters, ensuring alternate communication routes. This organization scheme results in the average communication “pathlengths” between brokers that increase logarithmically with geometric increases in network size, as opposed to exponential increases in uncontrolled settings.

NaradaBrokering supports dynamic broker and link additions and removals. While adding new brokers and links, it implements a broker organization protocol to avoid an unstructured broker network which hampers the development of efficient routing strategies. This lets broker network to grow or shrink dynamically.

Each broker keeps a broker network map (BNM) of its own perspective to efficiently route the messages to their destinations with a near optimal algorithm [nb2]. Messages are routed only to those routers that have at least one subscription for that topic. This prevents unnecessary message traffic on the system. Messages are also duplicated on brokers when they need to be sent to more than one hop. This is similar to multicast routing and saves significant bandwidth. Moreover, messages are routed only to the intended destinations and they are prevented from being routed back to the producers.

NaradaBrokering has a flexible transport mechanism [nb-transport]. Its layered architecture supports addition of new protocols easily. In addition, when a message traverses through broker network, it can go through different transport links in different parts of the system. A message can be transported over HTTP while traversing a firewall but later TCP or UDP can be used to deliver it to its final destinations. Therefore, it provides a convenient framework to go through firewalls.

Another important feature of NaradaBrokering is the performance monitoring infrastructure [nb-perf]. The performance of the links among brokers is monitored and problems are reported on real-time. In addition, this information is used to route messages through best paths.

As we have mentioned previously, publish/subscribe systems in general and NaradaBrokering in particular is not designed to deliver real-time audio and video streams. Therefore, we have made some additions to better support audio and video transfer. We have added an unreliable transport protocol (UDP). We have also added a compact message type which adds minimum headers to packages, RTPEvent. In addition, we have also implemented proxies for legacy RTP clients and multicast groups. Our initial testing shows that NaradaBrokering brokers can deliver significant performance. One broker can handle up to 400 high bandwidth video streams and more than 1000 audio streams [nb-conf].

6 GLOBALMMCS ARCHITECTURE

GlobalMMCS is a scalable service-oriented videoconferencing system. It utilizes the NaradaBrokering event brokering system as the media and content distribution network. In addition to scalability, it aims to provide universal accessibility by going through firewalls, NATs and proxies. It also supports a diverse set of clients by providing customized services. Moreover, it tries to provide an easy-to-use and maintain framework for developers and users. Many parts of this system is developed and used for experimental purposes.

There are three main components of this architecture: media and content distribution network, media processing unit and meeting management unit. In the previous section, we have already covered the media and content distribution network. In this section, before covering the other components of this system, first we will discuss the messaging and service distribution model.

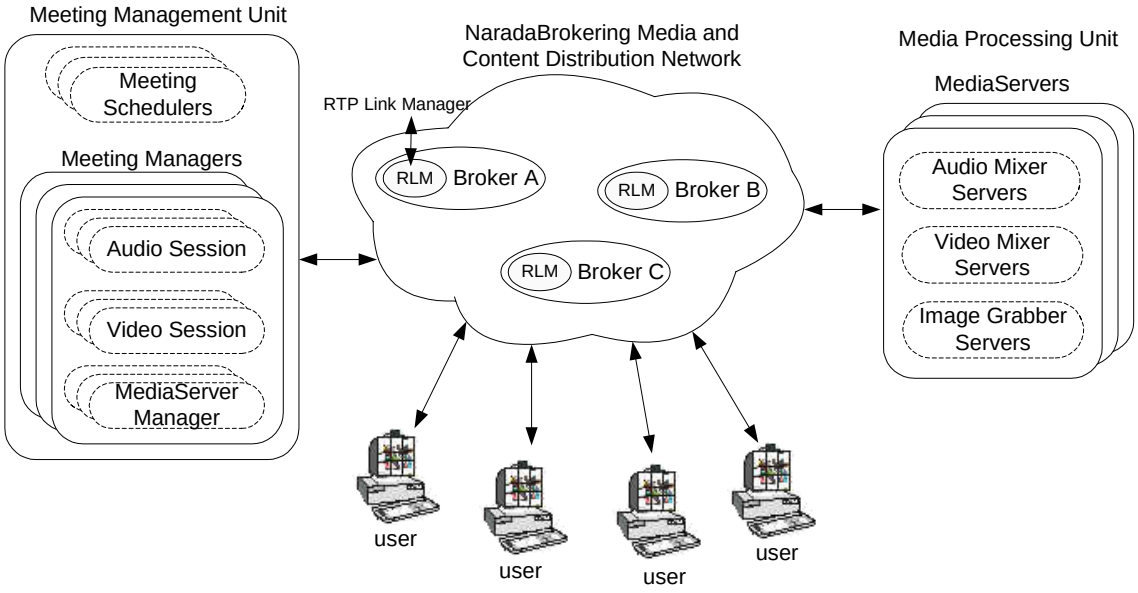

Figure 1 GlobalMMCS Architecture

6.1 Messaging Among System Components

In addition to audio/video delivery, we use NaradaBrokering-JMS publish/subscribe system to distribute the control messages exchanged among various components in the system. This simplifies building a scalable solution, since messages can be delivered to multiple destinations without explicit knowledge of the publisher. Service providers can

be added dynamically. Moreover, it provides location independence for each component, since a component is only connected to one broker and it exchanges all its data and media messages through this broker. In addition, using the same middleware for both data and media delivery reduces the overall system complexity considerably.

As we know, JMS provides a group communication medium. It uses topics as the group address. When a message is published on a topic, all subscribers of that topic receive that message. In our system, while some messages are sent to a group of destinations, some others are destined to one target. Therefore, an efficient and scalable message exchange mechanism should be designed among system components. Messages should only be delivered to intended destinations. In addition, topics should be organized in an orderly fashion.

First, we will examine the various messaging types that take place in our system. Then we will provide the topic naming convention to handle these messaging types.

6.1.2 Messaging Semantics

In our system, there are three different messaging types:

1. **Request/Response messaging**: This messaging semantic is used when a consumer requests a service from a service provider in the system. It sends a request message to the service provider to execute a service. The service provider processes the received message and sends a response message back to the sender. Since both the request and response messages are destined to one entity, it is important not to deliver these messages to unrelated components. Therefore, all service providers and consumers should have unique topics to receive messages destined to them only.
2. **Group messaging**: This messaging semantic is used when an entity wants to send a message to a group of entities in the system. It publishes a message to a shared topic and all group members receive it. In some cases, receiving components send a response message back to the sender. In some other cases, no response message is assumed. There are two types of applications of this messaging semantic in our system. First one is to use when discovering service providers. An entity sends a request message to the group address of a group of service providers. Then, each one of them sends a reply message including the information asked. Another application is to execute a service on a group of service providers. In this case, an entity sends a service execution request message to the group address, and all service providers belonging to that group execute that service.
3. **Event based messaging**: Event based messaging is used when an entity wants to receive messages from another entity regarding the events happening on that component during a period of time, such as over the course of a meeting. All interested entities subscribe to the event topic, and receive messages as the publisher posts them. A typical application of this event based messaging in our system is to deliver events related to audio and video streams to the participants of a meeting. All participants subscribe to the event topic and monitoring service publishes the events as they happen.

6.1.3 Topic Naming Conventions

To meet the requirements of the messaging semantics explained above, two types of topics are needed; group topics and unique component topics. We use a string based directory style topic naming convention to create topic names in an orderly and easy to understand fashion. All topic names start with a common root. We use our project name as the root name *GlobalMMCS*. But it is possible for an institution to change this name to another and all topic names change accordingly. This also lets installing more than one copy of this system on the same broker network. Group topic names are constructed by adding the component name to the root by separating with a forward slash. Groups are formed by the multiple instances of the same components. For example, all instances of MediaServers running in the system belong to the same group.

- GlobalMMCS/MeetingManager
- GlobalMMCS/AudioSession
- GlobalMMCS/VideoSession
- GlobalMMCS/ MediaServer
- GlobalMMCS/RtpLinkManager

These strings are used as the component group addresses. For example, all AudioSession objects listen on *GlobalMMCS/AudioSession* topic to receive messages which are destined to all AudioSession objects. Similarly, all other objects listen on their group addresses to receive group messages.

Unique component topic names are constructed by adding a unique id to these component group addresses:

- GlobalMMCS/AudioSession/<sessionID>
- GlobalMMCS/VideoSession/<sessionID>
- GlobalMMCS/ MediaServer/<serverID>

- GlobalMMCS/RtpLinkManager/<brokerID>

These unique topic names are used to communicate directly with a component. The messages sent to these topics only received by the component which has that id. When an instance of a component is initiated, it gets an id from the broker it is connected. Then it constructs its private topic name by following the above structure and starts listening on that topic for the messages destined to it. In addition to using the component id for constructing a private topic name, this id is also used to identify components from others in the system.

One of the additions which we made to NaradaBrokering is the mechanism to generate unique ids on time and space. A unique id generator runs in every broker and it can generate an id for every millisecond. This id will be unique for 557 years. Each broker generates unique ids without interacting with any other broker. Therefore, it is quite fast. Each id is 8 bytes.

Sometimes a component communicates with many different components; in that case, we use extra one more layer to distinguish these communication channels:

- GlobalMMCS/AudioSession/<sessionID>/RtpLinkManager
- GlobalMMCS/AudioSession/<sessionID>/AudioMixerServer
- GlobalMMCS/AudioSession/<sessionID>/RtpEventMonitor

In the above example, an AudioSession component communicates with three different entities: RtpLinkManager, AudioMixerServer and RtpEventMonitor. It uses different topics for each component. Using different topics simplifies logging and detecting the problems. It also simplifies developing codes to handle various types of messages exchanged with each component.

With this naming convention, we provide a unified mechanism to generate group and individual component topic names. It is easy to understand and debug.

6.2 Service Distribution Model

In our system, we support multiple copies of the same service providers in a distributed fashion. Since, there are a number of different service providers in our system, it would be better to have a unified framework for distributing the service providers. We assume that distributed copies should be able to run both in a local network and in geographically distant locations with different network connections.

As we have mentioned above, each service provider and the consumer is assigned a unique id. This id is used both to identify an instance of this component from others and to generate its unique topic name to communicate with others in the system. A service provider listens on two topics. One is the service provider group topic on which it receives messages destined to all service providers. Another is its private topic on which it receives messages sent only to itself.

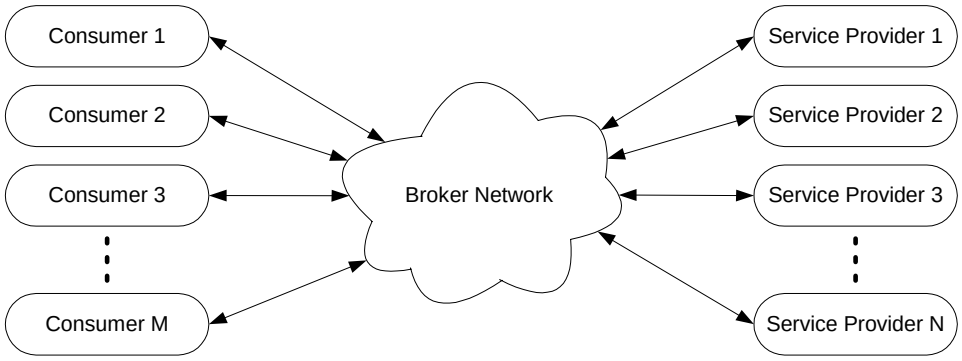

Figure 2 Service distribution model

6.2.1 Service Discovery

Instead of using a centralized service registry for announcing and discovering services, we use a distributed dynamic mechanism. One problem with centralized registry is the failure susceptibility of this approach. Another difficulty is that since in our system the status of the service providers changes dynamically, it is not reasonable to update a centralized registry frequently.

In our approach, a consumer sends an *Inquiry* message to the service provider group address. In this message, it includes its own topic name, so that service providers can send the response message back to it only. When service providers receive this message, they respond by sending a *ServiceDescription* message, in which they include the current status of that service provider. The information provided in this *ServiceDescription* message depends on the nature of the service being provided. But it must be helpful for the consumer to decide from which service provider to ask for the service. The consumer waits for a period of time for responses to arrive, and evaluates the received messages. Since a consumer does not know the current number of the service providers in the system, after waiting for a while it assumes that it received responses from all the service providers.

6.2.2 Service Selection

When a consumer receives the *ServiceDescription* messages from service providers, it compares the service providers according to the service selection criteria set by user. This criteria can be as simple as checking the CPU loads on host machines and choosing the least loaded one or it can take into account more information and complicated logic. For example, users can be given an option to set the preferences over the geographical location of the service providers. This can be particularly useful for systems that are deployed worldwide. This policy can be set by a configuration file to provide more flexibility.

6.2.3 Service Execution

When the consumer selects the service provider on which it intends to run its service, it sends a request message to the service provider for the execution of the service. If the service provider can handle this request, it sends an *Ok* message. Otherwise, it sends a *Fail* message. In the case of failure, the consumer either starts this process from the beginning or tries the second best option. A service can be terminated by the consumer by sending a *Stop* message.

In our system, a service is usually provided for a period of time, such as during a meeting. Therefore, the consumer and the service provider should be aware of each others continues existence during this time. Each of them sends periodic *KeepAlive* messages to the other. If either of them fails to receive a number of *KeepAlive* messages from the other, it assumes that the other party is dead. If the consumer is assumed dead, then the service provider deletes that service. If the service provider is assumed dead, then consumer looks for another alternative.

In our system, each service provider is totally independent of other service providers. Namely, service providers do not share any resources. Therefore, there is no need to coordinate the service providers among themselves. This simplifies the distribution and management of service providers significantly.

6.2.4 Advantages of this service distribution model:

- **Fault tolerance:** There is no single point of failure in the system. Even though some components may fail, others continue to provide services.
- **Scalability:** This model provides a scalable solution. There is no limit on the number of consumers to support as long as we have services to provide them. The fact that initially a consumer sends a message to all service providers, and they all respond back to the consumer, limits the number of the supported service providers. However, this can be eliminated by limiting the number of service providers who respond to specific inquiry messages. This selection can be based on the location of the service providers or some other criteria depending on the nature of the services provided. For example, already fully loaded service providers might ignore these inquiry messages.
- **Location independence:** All service providers are totally independent of other service providers and all consumers are also independent of other consumers. Therefore, a service provider or a consumer can run anywhere as long as they are connected to a broker.

6.3 Media Processing Units

We provide media processing services at server side to support a diverse set of clients. Some clients have high network bandwidth and advanced processing and display capacity. They can receive, process and display multiple concurrent audio and video streams. Therefore, they can receive all audio and video streams in a meeting. Some other clients have limited network bandwidth, processing and display capacity. Either they can not receive multiple audio and video streams or they can not process and display them. Therefore, server side components should generate combined streams for them. The services which we have implemented include audio mixing, video mixing and image grabbing. We also have an RTP stream monitoring service. All these services require real-time processing and usually high computing resources.

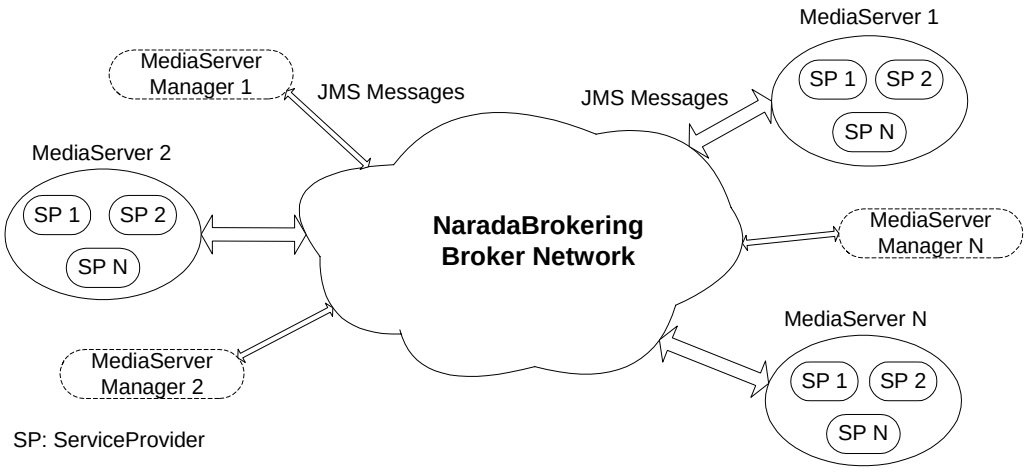

Figure 3 Media Processing Framework

Media processing framework (Figure 3) is designed to support addition and removal of new computing resources dynamically. A server container, MediaServer, runs in every machine that is dedicated for media processing. It acts as a factory for the service providers. It starts and stops them. In addition, it advertises these service providers and reports the status information regarding the load on that machine. All service providers implement the interface required by the server container to be able to run inside. Each MediaServer is independent of other MediaServers and new ones can be added dynamically.

Currently, there are three types of media processing service providers: AudioMixerServer, VideoMixerServer, and ImageGrabberServer. More service providers can be added by following the guidelines and implementing the relevant interfaces. These service providers can either be started from command line when starting the service container, or they can be started by using the MediaServerManagers. MediaServerManagers implement the semantics to talk to MediaServers.

6.3.1 Audio Mixing

AudioMixerServer provides audio mixing services, AudioMixerSession, for a meeting. An AudioMixerServer can have any number of audio mixers as long as the host machine can handle. Each speaker is added to the mixer as they join the meeting, and special mixed streams are constructed for them. Audio mixer receives the streams from the broker network and publishes back the mixed streams on the broker network. Clients receive the mixed streams by subscribing to the mixed stream topics.

While some audio codecs are computing intensive, some others are not. Therefore the computing resources needed for audio mixing change accordingly. Audio mixing units need to have prompt access to CPU when they need to process received packages. Otherwise, some audio packages can be dropped and result in the breaks in audio communications. Therefore, the load on audio mixing machines should be kept at as low as possible.

Number of audio mixers	CPU usage %	Memory usage MB	Quality
5	12	36	No loss
10	24	55	No loss
15	34	73	No loss
20	46	93	Some loss, Negligible

Table 1 Audio mixer performance test

We have tested the performance of an AudioMixerServer for different number of mixers on it. There were 6 speakers in each mixer. Two of these speakers were continually talking and the rest of them were silent. There were also one more audio stream constructed which had the mixed stream of all speakers. Therefore, 6 streams were coming into the mixer and 7 streams were going out. All streams were 64kbps ULAW. Mixers were receiving the streams from a broker and publishing the output streams back on the broker. The machine that was hosting the mixer server was a winXP machine with 512 MB memory and 2.5 GHz Intel Pentium 4 CPU. The broker was running on another machine in the same subnet.

We can say that the number of speakers in the test in a mixer is not less than the average number of speakers in meetings. Usually there is one active speaker in a session and there are less than 6 speakers. Therefore, we assume that this setting represents at least average meetings. Table 1 shows that a machine can support around 20 mixing sessions. But we should note that, in this test all streams are ULAW. This is not a computing intensive codec. When we had the same test with another more computing intensive codec, G.723, one machine supported only 5 mixing sessions. Therefore, on the average we can say one machine may support around 10 mixing sessions.

6.3.2 Video Mixing

There are a number of ways to mix multiple video streams into one video stream. One option is to implement a picture-in-picture mechanism. One stream is dedicated as the main stream and it is placed in the background of the full picture. Other streams are imposed over this stream in relatively small sizes. Another option is to place the main stream in a relatively larger area than other streams. For example, if the picture area is divided into 9 equal regions, main one can take 4 consecutive regions and remaining regions can be filled with other streams. In our case, we choose a simpler way of video mixing. We divide the picture area into four equal regions and place a video stream into each region. This lets a low end client to display four different video streams by receiving only one stream. VideoMixerServer can start any number of VideoMixers. Each video mixer can mix up to 4 video streams. Therefore, in large meetings more than one video mixing can be performed.

Video mixing is a computing intensive process. One video mixer decodes four received video streams and encodes one video stream as the output. In Table 2, it shows that a Linux machine with 1 GB memory and 1.8GHz Dual Intel Xeon CPU, can serve 3 video streams comfortably and 4 at maximum. Therefore, video mixing is a very computing intensive process. In this test, we used the same incoming video stream for all mixers. The incoming video stream was an H.261 stream with an average bandwidth of 150kbps. The mixed video stream was an H.263 stream with 18fps.

Number of video mixers	CPU usage %	Memory usage (MB)
1	20	42
2	42	54
3	68	68
4	94	80

Table 2 Video mixer performance test

6.3.3 Image Grabbing

The purpose of image grabbing is to provide users with a meaningful video stream list in a session. Without the snapshots of the video streams, users are often confused to choose the right video stream for them. Snapshots provide a user friendly environment by helping them to make informed decisions about the video streams they want to receive. Therefore, it saves a lot of frustration and time by eliminating the need for trying multiple video streams before finding the right one.

An image grabber is started for each video stream in a meeting. This image grabber subscribes to a video stream and gets the snapshots of this stream regularly. It first decodes the stream, then reduces its size to save CPU time when encoding and transferring the image. Then it encodes the picture in JPEG format. Either the newly constructed image can be saved in a file and served by a web server, or published on the broker network and accessed by subscribing to relevant topics.

Number of image grabbers	CPU usage %	Memory usage (MB)
10	15	66
20	35	110
30	50	148
40	60	192
50	70	232

Table 3 Image grabber performance test

Image grabbing is also a computing intensive task. Each image grabbing includes decoding, resizing and encoding of a video stream. Though, resizing and encoding do not have to be done continually. They can only be performed when it is time to get the snapshot. Table 3 shows the performance tests for image grabbers. All image grabbers subscribed to the same video stream on a broker. That video stream was in H.261 format with an average bandwidth of 150kbps. Image grabbers saved a snapshot every 60sec to the disk in JPEG format. The host machine was a Linux machine with

1 GB memory and 1.8GHz Dual Intel Xeon CPU. Although the number of video streams and the format of the streams change, if we assume that meeting has 10 video streams on the average, Table 3 shows that one machine can serve 5 meetings.

6.3.4 RTP Stream Monitoring

Stream monitoring service monitors the status of audio and video streams in a meeting, and publishes the events happening on dedicated topics. The entities interested in these events subscribe to these topics and receive them as the monitoring service publishes them. For example, all participants in a meeting subscribe to audio and video stream events to receive them. Currently, there are four types of events: StreamReceivedEvent, ByeEvent, ActiveToPassiveEvent and PassiveToActiveEvent. Each of these events gives information regarding a particular steam. These events also provide information about the identity of senders of these streams.

Contrary to other media processing services, stream monitoring is not implemented as a stand alone application. Instead, audio stream monitoring is implemented along with audio mixing service and video stream monitoring is implemented along with image grabbing service. Since all audio streams in a meeting are received by the audio mixer, and all video streams are received by image grabbers, we embedded the stream monitoring services into them to avoid extra audio and video stream delivery.

6.3.5 Media Processing Service Distribution

We use the previously explained service distribution model to distribute the media processing tasks. MediaServerManager implements the logic to talk to server containers and also for selecting the best available service providers. Currently, we use simple distribution logic for small settings. But we are working on more complete scalable algorithms. We plan to include prediction of necessary resources for a meeting and schedule accordingly. Users can be asked to provide more information about the meeting when they are scheduling. For example, they can provide the expected number of participants. This can be very helpful when scheduling audio mixers and image grabbers.

6.4 Meeting Management Unit

Meeting management unit handles starting/stopping/modifying videoconferencing sessions. It also manages the media processing unit resources by using MediaServerManagers. In addition, it manages participant joins and leaves.

AudioSession and VideoSession objects are lightweight components, which manage audio and video meetings, respectively. This management includes two main functions. First one is to manage the topics used for a meeting. They keep the list of users and the topics they publish their media. The second one is to provide session management services to participants, such as user joins and leaves. While handling these requests, they usually talk to other system components, such as media processing units and RTPLinkManagers. MediaServerManagers are used by MeetingManagers to locate and to start/stop media processing servers. On the other hand, MeetingSchedulers are used to initiate and to end AudioSession and VideoSession instances. MeetingSchedulers can run either as independent applications or as embedded components in web servers. When they are used with web servers, an administrator or a privileged user initiates meetings through a web browser.

Although, session management components are lightweight entities and they can handle a large number of concurrent users, we still distribute AudioSession and VideoSession objects to provide fault tolerance. We use the service distribution model outlined in the previous section. MeetingManagers act as service providers and MeetingSchedulers act as consumers.

Here we explain the message exchanges that take place when creating a videoconferencing session. A MeetingScheduler sends an Inquiry message to MeetingManagers in the system. After receiving the responses, it selects a MeetingManager to ask for the service. It sends two request messages to the selected manager: CreatAudioSession and CreateVideoSession. This MeetingManager uses a MediaServerManager to locate an AudioMixerServer and an ImageGrabberServer to start the AudioMixerSession and the ImageGrabberSession, respectively. Then, it starts an AudioSession instance while providing the selected AudioMixerServer. This AudioSession object asks the given AudioMixerServer to start an AudioMixerSession to be used during this meeting. MeetingManager also initiate a VideoSession instance while providing the identified ImageGrabberServer. This VideoSession also asks the given ImageGrabberServer to start an ImageGrabberSession to be used during this meeting. This completes the initialization of the session. Users can join the session by sending Join messages directly to AudioSession and VideoSession components. A VideoMixer can also be added by exchanging messages with the VideoSession object. We should also note that MeetingManager accesses MediaServerManager directly by calling its methods.

Here we also would like to explain briefly the messaging that takes place when users join meetings. When a speaker joins an AudioSession, a topic number is assigned for this user to publish its audio stream. Another topic number is also assigned to publish the mixed audio stream for this user by the audio mixer component. This user is also added to the AudioMixerSession. The mixer constructs a new stream for this user and publishes it in the given topic number. The interaction between the AudioSession and AudioMixerSession components are transparent to the user. If the joining user is a listener, in that case it is only given the mixed stream topic number to receive the audio of all speakers in the session. Since it will not publish any audio, it is neither assigned a topic number, nor added to the mixer.

When a speaker joins a VideoSession, it is assigned a topic number to publish its video stream. Then, an image grabber is also started to construct the snapshots of its video stream. This user is also given the list of available video streams in the meeting. He/she can subscribe to these streams by sending subscribe/unsubscribe messages to the VideoSession object.

6.5 Scalability of GlobalMMCS

We believe that our approach provides a very flexible and scalable solution to serve high number of participants. Additional media processing units can be added easily to provide more computing resources. More brokers can also be added easily to support the distribution of high number of audio and video streams. When this system is deployed globally, clusters of media processing units can be deployed worldwide to handle the needs of tens of thousands of users. This would require intelligent algorithms to locate the right media processing units and distribute the load intelligently.

Although we have not tested our system in large scale settings, here we would like to discuss some large scale cases based on the results we have presented above. First, we envision a university graduation ceremony broadcasted to families and students who could not join the ceremony. Let's assume that 5 thousand users want to access it over Internet. These users might have various capabilities. Some of them only want to listen the ceremony through cell phones and some others want to watch multiple video streams on their PCs. We can assume that multiple video streams are transmitted from various angles. Let's assume that there are 10 different video streams broadcasted. Probably there will be only one audio stream transmitted. Therefore, audio mixing will not be needed. One or two mixed video streams can be provided. An image grabber can be started for each video stream. This setting requires very little resources from media processing unit. One machine for video mixers and another for image grabbers can be enough. The real challenge will be on brokering network to deliver the audio and video streams to those many users. Fortunately, the limited number of audio and video streams simplifies the delivery task significantly. The maximum number of video streams transmitted from one broker to another can not be more than the total number of streams on the session. That is 11 streams. Therefore, a broker can receive 11 streams at the maximum and deliver them to users. Our previous tests show that one broker can send one video stream up to 400 users. If we assume that each broker can serve 200 users comfortably, then 25 brokers would be needed to serve 5000 users. One very important aspect of this problem is the distribution of users among brokers. Users can be congested on some brokers if no action is taken to distribute them evenly. To circumvent this problem, first of all, brokers should be deployed according to the user distribution. Then users should be directed to right brokers when they join the session. The decision to locate a broker for a user should take into account both the load of a broker and the location of the broker.

Next we consider an online university that is providing classes to a thousand students. We assume that there can be as many as 50 concurrent classes each having 20 students. Each class can have a few video streams and a few audio streams. One audio stream is the audio of the teacher and the others can be those students who are promoted to speaker roles to ask questions or make comments. Therefore, an audio mixer is necessary for each session. For some sessions a video mixer can be provided. Image grabbing would be necessary for all video streams. Let's assume that there are 150 audio and 150 video streams in total. 50 audio mixers require at least 3-4 machines. 150 image grabbers also require 3-4 machines. If there are 20 video mixers, that would also require 6-8 machines. Therefore, 12-15 machines would be required for media processing. However, media delivery would be more challenging. Since there are 300 streams in total, in the worst case, if all streams are routed on a link between two brokers in the network at one point, it would overwhelm the link and cause congestion. That would result in serious package losses. Therefore, when routing paths are calculated, the loads on the links and brokers must be taken into account. Although the current routing algorithm in NaradaBrokering does not take into account the dynamic changes in the loads, new algorithms are being developed to monitor the loads on the links and route accordingly. Since there are more streams routed inside the network, in this case, one broker would be able to serve less number of clients. If we assume that each broker serve 100 users, then at least 10 brokers would be necessary to serve 1000 participants.

7 CONCLUSION

In this paper, we proposed a service oriented architecture to implement scalable videoconferencing systems. This system utilizes a publish/subscribe messaging middleware to transfer both multimedia and data traffic. The key to the scalability of this architecture is the usage of NaradaBrokering event broker network as the middleware and also the

implementation of a service oriented component distribution mechanism. In this system, new resources can be added transparently. We have gathered initial performance tests which show that this approach can deliver significant performance. We are still working on developing algorithms that would allow global distribution of various components. We are also working on intelligent routing algorithms that would take into account the dynamic loads on the broker network

8 REFERENCE

[h323] ITU-T Recommendation H.323, "Packet based multimedia communication systems", Geneva, Switzerland, Feb. 1998.

[mult-evol] K. Almeroth, "The Evolution of Multicast: From the MBone to Inter-Domain Multicast to Internet2 Deployment", IEEE Network, Jan 2000, Volume 14.

[sip] J. Rosenberg et al., "SIP: Session Initiation Protocol", RFC 3261, Internet Engineering Task Force, June 2002, http://www.ietf.org/rfc/rfc3261.txt

[ps-faces] P. Th. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. ACM Computing Surveys, Vol. 35, No. 2, June 2003, pp. 114–131.

[ps-evol] R. Baldoni, M. Contenti, A. Virgillito. The Evolution of Publish/Subscribe Communication Systems. "Future Directions of Distributed Computing", Springer Verlag LNCS Vol. 2584, 2003

[Tapestry] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. Katz, and J. Kubiatowicz. Tapestry: An infrastructure for fault-tolerant wide-area location and routing. Technical Report UCB/CSD-01-1141, University of California at Berkeley, Computer Science Division, April 2001.

[bayeux] S. Q. Zhuang, B. Y. Zhao, A. D. Joseph, R. Katz, and J. Kubiatowicz. Bayeux: An architecture for scalable and fault-tolerant wide-area data dissemination. In 11th Int. Workshop on Network and Operating Systems Support for Digital Audio and Video, 2001.

[end-system] Yang Hu Chu, Sanjay G. Rao, and Hui Zhang. A case for end system multicast. In Proc. ACM SIG-METRICS Conference (SIGMETRICS '00), June 2000.

[nb1] Shrideep Pallickara and Geoffrey 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.

[nb2] Geoffrey Fox and Shrideep Pallickara. An Event Service to Support Grid Computational Environments. Journal of Concurrency and Computation: Practice & Experience. Special Issue on Grid Computing Environments. Volume 14(13-15) pp 1097-1129.

[garnet] Geoffrey Fox et al. "Grid Services For Earthquake Science". Concurrency & Computation: Practice and Experience. Special Issue on Grid Computing Environments. Volume 14:371-393.

[nb-transport] Shrideep Pallickara, Geoffrey Fox, John Yin, Gurhan Gunduz, Hongbin Liu, Ahmet Uyar, Mustafa Varank. A Transport Framework for Distributed Brokering Systems. Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications. (PDPTA'03).Volume II pp 772-778.

[nb-perf] Gurhan Gunduz, Shrideep Pallickara and Geoffrey Fox. A Framework for Aggregating Network Performance in Distributed Brokering Systems. Proceedings of the 9th International Conference on Computer, Communication and Control Technologies.Volume IV pp 57-63.

[jms] Mark Happner, Rich Burridge and Rahul Sharma. Sun Microsystems. Java Message Service Specification. 2000. http://java.sun.com/products/jms.

[nb-conf] Ahmet Uyar, Shrideep Pallickara, Geoffrey Fox, "Towards an Architecture for Audio/Video Conferencing in Distributed Brokering Systems", The proceedings of The 2003 International Conference on Communications in Computing, June 23 - 26, Las Vegas, Nevada, USA.

[xgsp] Design and Implementation of a collaboration Web-services system: By Wenjun Wu, Geoffrey Fox, Hasan Bulut, Ahmet Uyar, Harun Altay, to appear in Journal of Neural, Parallel & Scientific Computations.

[globalmmcs] Global Multimedia Collaboration System. By Geoffrey Fox, Wenjun Wu, Ahmet Uyar, Hasan Bulut, Shrideep Pallickara. Concurrency And Computation:Paractice and Experience. 2004; 16:441-447