Integrated Streaming Service Architecture: A Streaming Framework Compatible with Global Multimedia Databases*

Sungyoung Lee[1], Byungsoo Jeong[1]
[1] Dept. of Computer Engineering, Kyung Hee University, Seoul, Korea
{sylee@oslab.kyunghee.ac.kr, jeong@khu.ac.kr}

Abstract This paper describes a design for an integrated streaming framework that can provide an environment to develop multimedia streaming applications under heterogeneous distributed systems. Our streaming framework was designed to extend the limits of existing streaming systems while offering easy interaction with other streaming systems, to support diverse media formats and networks, and to provide high level programming environment for streaming application developers. The framework is also compatible with global real-time multimedia DBMS (BeeHive) so that streaming media is efficiently retrieved, stored, and serviced using database systems.

Keywords: streaming, multimedia, audiovisual, real-time, database

1. Introduction

Streaming is a technology for multimedia communications that makes it possible to deliver and display multimedia (such as audio and video) data in real-time. By using streaming technology, users can access multimedia contents immediately upon demand without waiting for the whole file to be downloaded. It is well suited to the transmission of video data with real-time characteristics and high bandwidth communication requirements. It might be the only practical solution possible for live video streams.(It makes the transmission of video date with real-time characteristics and higher bandwidth communication requirements suitable.)

Numerous streaming systems existing, most systems do not consider important issues such as interoperating ability with other streaming systems, supporting diverse media formats and network interfaces. Thus they lack flexibility, extendibility, and network transparency. The increasing number of coding and compression standards has led to a situation where heterogeneity is a growing problem. If a server uses many different coding formats, clients must be able to decode these formats. In order to relieve streaming applications from interoperability and heterogeneity concerns, there should be a streaming framework which provides diverse audio and video CODEC and network interfaces.

Meanwhile, current streaming systems, either, simply provide users with multimedia files stored in hard disks of the streaming server, or store the information of the multimedia data in a relational database so that users can query on the information. Upon streaming, such a streaming system, which interconnects with the relational database, transmits multimedia data stored in the file system of the server. In other words, in such a system, the multimedia data is managed separately from its information or meta-data. Therefore, multimedia services such as retrieval and join operations during streaming can not be provided, since it still has the service constraints which file systems have.

In order to overcome the constraint, we also propose a Database Connector as an interconnection scheme between multimedia database, so called BeeHive under development in the University of Virginia, and ISSA(Integrated Streaming Service Architecture). Much research has been done in the area of multimedia database systems. To our knowledge, however, there is no commercial multimedia database that can be interconnected with any commercial streaming systems. So, as an interconnecting target database system, we selected BeeHive, a global real-time multimedia database system, whose source codes are accessible. BeeHive offers features such as real-time, fault tolerance, quality of service for audio and video, and security dimensions. ISSA, on the other hand, is a streaming system framework which is easy-to-extend, able to run adaptively on different operating systems and networks, and supporting diverse audio/video media formats.

Through the interconnection between the streaming system and multimedia database, the information related to the media under play can be retrieved in various formats as users require while streaming. The users can also retrieve, join and stream the multimedia data according to their requirements. Diverse multimedia services can be provided through the interconnection because not only the information about the multimedia data but also the multimedia data itself are under the management of the database system. For this purpose, in the Database Connector proposed in this paper, transactions that can be processed in the multimedia database were defined, and interfaces for the transaction processing were defined and implemented. A movie database was chosen as an application area for the implementation, and Read, Write, Find and Play were defined as transaction primitives to store and manage Movie Objects.

The interoperable interface model between ISSA and BeeHive consists of Transaction Interface and Streaming Interface. The Transaction Interface deals with a series of functions through which ISSA

* Supported in part by contract IJRP-9803-6 from the Ministry of Information and Communication of Korea

clients request for transactions of BeeHive. The Streaming Interface provides a mechanism to stream media between an ISSA client and the server. An IPC(Inter-Process Communication) based message queue and a shared memory scheme are used for the message exchange between ISSA and BeeHive. In this paper, we present an approach to designing an integrated streaming framework which can overcome the limitations of existing streaming systems while providing diverse audio and video CODEC and the ability to run adaptively on different operating systems and networks.

This paper proceeds as follows. Section 2 provides an overview of existing streaming frameworks that are similar to our system. We describe the system architecture of our integrated streaming framework in Section 3. We introduce the data base connector as a streaming framework compatible with multimedia databases in Section 4. We show the performance result of the database connector as a result of compatible with global real-time multimedia database (BeeHive) and ISSA in section 5 and finally states our conclusion in Section 6.

2. Related Work

This section briefly describes existing streaming systems that are related with our work. Most of the systems mentioned below are the programming environments that can efficiently manipulate real-time multimedia streams like audio and video. CMT(Continuous Media Toolkit) is a multimedia programming toolkit developed at MIT, which provides a programming environment for rapid development of continuous media applications [1]. However, CMT is hard to be easily extended, since its internal structure is very complicated and flat; furthermore, it does not follow the object-oriented programming paradigm. VuSystem is a programming environment developed at MIT, which facilitates the development of compute-intensive multimedia applications, combining intelligent media processing with traditional capture and display [2]. However, VuSystem does not provide RTP and Multicast, and its user interface has very limited functionalities. Furthermore, since VuSystem does not support media compression, it cannot guarantee QoS in the environment of heterogeneous networks with different bandwidths. KISS (Communication Infrastructure for Streaming Services) is a communication infrastructure for streaming services that allows the transparent integration of network service applications, and also adapts real-time content streams to network conditions and/or individual client requirements [3]. Although KISS provides network transparency for streaming application developers, it has the drawback of supporting only audio streams such as PCM and MPEG-1 audio layer 3, and it suffers from the performance overhead of NAP.

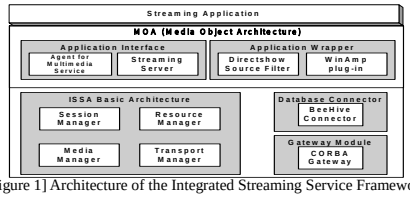
To facilitate the development of standard-based distributed multimedia streaming applications, the OMG (Object Management Group) has defined a CORBA-based specification for the control and management of A/V streams [7], based on the CORBA reference model [5]. OMG A/V streaming architecture is represented as a flow between two flow data endpoints. One endpoint acts as a source of the data and the other endpoint acts as a sink. In the OMG streaming architecture, the control and signaling operations pass through the GIOP/IIOP-path of the ORB, demarcated by the dashed box. By contrast, the data stream uses out-of-band streams, which can be implemented using protocols that are more suitable for multimedia streaming than IIOP. Washington University has developed the first freely available implementation of the OMG A/V streaming model using TAO [6], which is a real-time CORBA ORB that has been ported to most OS platforms [4]. However, the CORBA A/V streaming service based on TAO does not implement the OMG specification completely. It implements a simple MPEG video player that has session control functions only. Thus, it cannot be used as a streaming framework.

3. Integrated Streaming Framework

This section explains the basic architecture and functions of ISSA, the streaming system proposed in this paper, and the BeeHive multimedia database system. As shown in Figure 1, the ISSA framework model consists of the Streaming Application, MOA(Media Object Architecture)[9] which is the interface between ISSA and the Streaming Application, Database Connector, Gateway Module, and ISSA. MOA further consists of the Application Interface and the Application Wrapper. The Application Interface is divided into two parts, AMS(Agent for Multimedia Service) and Streaming Server. The AMS integrates and distributes the information of the contents or sessions scattered in each streaming server. The Streaming Server serves as a media transmission function through the integration of RTSP(Real-Time Streaming Protocol)[10] modules, and plays the role of informing the AMS of the contents it has or informing existing sessions. The Application Wrapper is composed of Directshow Source Filter and Winamp plug-ins. The Directshow Source Filter is used for interconnection with MS Windows Media Player, and Winamp plug-ins are used for the MP3 streaming services. The Database Connector has the BeeHive Connector for the interconnection with the BeeHive real-time multimedia database, and the Gateway Module consists of the CORBA Gateway for interconnection with CORBA[11][12][13].

The basic architecture of ISSA, the core of the framework, is composed of Session, Transport, Media and Resource Manager. The Session Manager is in charge of the session control of the RTSP-based multimedia streaming service, and is configured to support unicasting and multicasting. It also supports the interface for database transaction requests, and SCP(Session Control Protocol) for the distribution and sharing of the content informations. SCP is defined for the communication between AMS and the Streaming Server, and is used when the Streaming Server needs an update to add new contents, or when the server needs to transmit its contents to the AMS as it is newly operated. The Transport Manager is in charge of transmitting multimedia data using TCP, UDP and RTP(Real-time Transport Protocol) [8]/UDP protocols, and also monitors the network status using the RTCP(Real-Time

Control protocol) protocol. The Media Manager is in charge of the media encoding/decoding, and it supports MPEG-1, MPEG-2, ASF and MP3. The Resource Manager provides specification of QoS, mapping, monitoring and controlling functions in the streaming system, and performs memory buffer management and thread scheduling.



[Figure 1] Architecture of the Integrated Streaming Service Framework

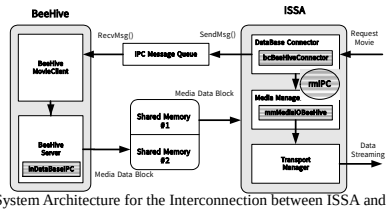
The BeeHive is a real-time multimedia database system which supports diverse multimedia functions on the SHORE file system, developed in University of Wisconsin. It furnishes real-time support, QoS for audio/video, fault-tolerance and security, and it is also adequate to apply the interconnection method, proposed in this paper, because it is an open system. It is composed of the Native BeeHive Sites, connection ports for the outside systems, and the interface for the interconnection between the BeeHive and the outside systems.

4. Database Connector

In this section, we describe about developing the Database Connector as an interconnection method between multimedia database, and the streaming framework. It is possible to support diverse and mature multimedia database services such as retrieval and join operation during the streaming if an interconnection method is provided in between streaming system and multimedia databases. The currently available interconnection schemes, however, have mainly used the file systems or the relational databases that are implemented with separated form of meta data, which deals with information of multimedia contents, and streaming data which deals with multimedia data itself. Consequently, existing interconnection mechanisms could not come up with many virtues of multimedia database services during the streaming operation. In order to resolve these drawbacks, we propose a novel scheme for an interconnection between streaming framework and multimedia database, called the Inter-Process Communication based Database connector, under the assumption that two systems are located in a same host. We define four transaction primitives; Read, Write, Find, Play, as well as (define) the interface for transactions that are implemented based on the plug-in, which, in consequence, can extend to other multimedia databases that will come for some later years.

There are two interfaces. One is the Transaction Interface which processes transactions needed for the interconnection between the streaming system and the multimedia database, and the other is the Streaming Interface which transmits the requested multimedia data. Four transaction primitives, Read, Write, Find, Play, are defined in the Transaction Interface, where each transaction requests from users are transformed into the transaction used in the multimedia database system. The Streaming Interface sends the resulting multimedia data to the user. The Database Connector interconnects the Streaming Server to the multimedia database. The IPC message queue is used to transactions, and the IPC shared memory is used to exchange meta-data or multimedia data.

The overall structure for the interconnection between ISSA and BeeHive is as in Figure 2. The ISSA module is composed of the BeeHive Connector, Media Manager and Transport Manager. The BeeHive module consists of the BeeHive Server and a Movie Client which manages the movie database. The IPC semaphores, same number as the number of memory blocks, is used in order to resolve the synchronization problem which may occur when two systems use the IPC shared memory blocks.



[Figure 2] System Architecture for the Interconnection between ISSA and BeeHive

Figure 2 shows the proposed IPC-based interconnection model. Once the Database Connector module receives a movie request from a client, BeeHive interconnecting module, bcBeeHiveConnector, requests the media stored in the message queue. Then, BeeHive Movie Client sends the contents of the message queue to the BeeHive Server, which, in turn, extracts the stored media and loads it into the shared memory using the inDataBaseIPC, the plugin module for the shared memory access. Once the media data is loaded in the shared memory, mmMediaIOBeeHive module in the Media Manager sends the data to the Transport Manager, which, in turn, transmits the data to the client.

As shown in Figure 2, in the ISSA side, the classes implemented in order to interconnect two systems are bcBeeHiveConnector, IPC shared memory, message queue, rmIPC for semaphores. In the BeeHive side, IPC shared memory, message queue and inDataBaseIPC class, which accesses to semaphores and database to fetch media data, were implemented. The bcBeeHiveConnector class initializes the Database Connector and generates and removes sessions with BeeHive. The mmMediaIOBeeHive class, inherited from the mmMediaIO class which selects and reads media source, is used to open and read the media data stored in BeeHive upon interconnection. The rmIPC class generates and controls the IPC shared memory and semaphores, and contains a method which sends messages to BeeHive. The inDataBaseIPC class is in charge of all the database operations when BeeHive interconnects with the streaming system. It accesses to the IPC shared memory from the database side and extracts media blocks from the database.

Figure 3 shows a pseudocode which describes how the modules of ISSA and BeeHive interoperate when multimedia data streaming is requested from the ISSA client. Messages delivered using the message queue are classified into 6 types according to the contents of the Command field. Message r generates a session, assigns a session ID, and fetches the information about the requested media. The session ID acquired from the message r is used to generate an ID and an address for the IPC shared memory and an ID for the IPC semaphore of which the object, in charge of ISSA's database IO, generates. Once a message r is transmitted to the database, the information about the multimedia data corresponding to the *database ID*, for example, the meta-data such as the title of the movie, name of the director and actors in case of movie database, can be acquired. Message *p* requests the media block stored in database for streaming. Once a message *p* is generated, the media block with the requested *Database ID* is extracted from the database and copied into the shared memory. This process continues until a message *q* is generated. Message *q* stops transmission of multimedia data. Once a message *q* is generated, the database system stops copying the media block into the shared memory. Message *l* shows the list of media currently stored in database. Message *s* stores multimedia data into database. Using the message *s*, a unique *database ID* is assigned to the media to be stored. Message *d* deletes the multimedia data from the database.

```
Line 1: ISSA Server receives the Multimedia Streaming Request  from the ISSA Clients
Line 2: Initialize the BeeHive Connector and Get the Message Queue address from the BeeHive MoiveClient
Line 3: Create mmMediaIOBeeHive Object
Line 4: Send r Message to the BeeHive MovieClient  // r message Creates session and requests the Media
        Information for the Multimedia Streaming  //
Line 5: Create Shared Memory and Semaphore
Line 6: Get Shared Memory address and Semaphore key Value
Line 7: Send p Message to the BeeHive MovieClient  // p message requests the invocation of streaming Start //
Line 8: While( Requested Media Block != End of Media Block )
Line 9: BeeHive Server copies Media Block to Shared Memory
Line 10: ISSA Server streams Media Block from Shared Memory
Line 11: Send q Message to the BeeHive MovieClient // q message requests streaming stop //
```

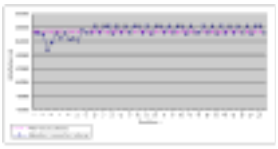
[Figure 3] Pseudocode showing operations between ISSA and BeeHive

The procedures down to Line 7 are self-explanatory. After the streaming start is requested, the BeeHive Server extracts the media data through the inDataBaseIPC object, and copies the first media data to the shared memory block, and then passes the control of the shared memory block to the mmMediaIOBeeHive. The mmMediaIOBeeHive starts streaming the media block from the shared memory. While the first media block is being streamed, the BeeHive Server copies the next media block to stream to the other shared memory block. When there is no more data for streaming in the shared memory block, the mmMediaIOBeeHive takes over the control of the next shared memory block to continue streaming. The process between the BeeHive Server and the mmMediaIOBeeHive object continues until the last media block is extracted by the BeeHive Server and streamed(Line 8,9,10). Once all the blocks of the selected media is transmitted to the client, the mmMediaIOBeeHive object creates the *q* message requesting the streaming to stop(Line 11).

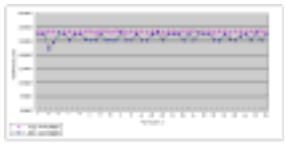
5. Performance Evaluation of Database Connector

We executed performance evaluation under the intranet/internet environment with 10 Mbps bandwidth. BeeHive and the ISSA server were operated on a SUN workstation with SunOS 5.5.1. The media data was stored splitted into blocks in 512 Kbyte size in order to reduce the system load and minimize the affect on the streaming service quality. The client was implemented using MicroSoft Windows Media Player 6.4 with the RTP source filter developed in ISSA. We evaluated the performance in two ways. First, we compared the RPC-based interconnection method with the proposed IPC-based interconnection method, and tested which method is more efficient for streaming multimedia data. Secondly, we compared streaming multimedia data stored in a file system with streaming media data stored in a multimedia database with IPC-based interconnection method, and tested data transmission rates (in seconds) in both methods to tell the differences.

We compared the performance of two cases, streaming media stored in file system and multimedia database, by testing the data transmission rate. This experiment was performed under both intranet and internet environments. We established a virtual experimental environment for more accurate comparison with other streaming systems. We also operated the streaming servers for both file system and database system on the same host at the same time, considering the fact that the network bandwidth changes in time. The MPEG-1 media encoded in 1392640 bits/sec was used for the experiment. Figures 4 and 5 show the data transmission rate(BytePerSecond) of the streaming server under the intranet environment(Experiment 1) and internet environment(Experiment 2), respectively.



[Figure 4] Experiment 1: the Average Transmission Rates of the Streaming Server for both File System and Multimedia Database under Intranet Environment



[Figure 5] Experiment 2: the Average Transmission Rates of the Streaming Server for both File System and Multimedia Database under Internet Environment

Through Experiments 1 and 2, we found that, in the case of streaming media stored in a file system, the streaming server supplies regular amount of data regardless of the lapse of time. In the case of streaming media stored in the BeeHive database using the IPC-based interconnection method, streaming did not become stable until few seconds from the beginning of the streaming passed, due to the initial delay to extract media blocks from the database. As shown in the Figures 12 and 13, in the case of interconnecting with multimedia database, the graphs change inconstantly. This phenomenon is caused by the time delay that occurs when the streaming server exchanges memory blocks to fetch media data. The change in the data transmission rate is not big enough to affect the streaming process. Therefore, we showed that the proposed streaming method using multimedia database system can provide multimedia database services without big loss of performance, compared with the file-based streaming method.

6. Summary

This paper describes an integrated streaming framework that integrates heterogeneous environments and work easily together with other streaming systems. Our streaming framework gives more robustness, flexibility, and extendibility than existing streaming systems since it is designed according to an object-oriented paradigm. It also supports diverse media formats and heterogeneous network environments. Simplified APIs by MOA provide efficient programming environment for the streaming application developer. Our streaming framework can also handle large amounts of media data through interworking with DBMS.

Futhermore, we proposed an IPC-based Database Connector as an interconnection module between the ISSA streaming system and BeeHive, a real-time multimedia database. The interconnection module consists of the transaction interface to process multimedia database transactions, and the streaming interface to stream media data stored in multimedia database. IPC-based message queue, shared memory and semaphores were implemented in order to deliver messages and data between two systems. From the results of the performance evaluation, we concluded that streaming method using multimedia database can provide multimedia services without big loss of QoS, compared to the file-based streaming method. Since the IPC interface module was implemented as a plugin, the proposed interconnection method is extensible to the other multimedia databases.

In the future, we plan to implement our system under diverse OS platforms, such as UNIX and MS-Windows, and to integrate it with BeeHive, a global real-time database system which is under development at the U. of Virginia. We also plan to reduce the performance overhead of our system as much as possible since we believe that performance is a major factor to guarantee QoS in multimedia systems.

7. References

[1] K. Mayer-Patel, and L. A. Rowe, "Design and Performance of the Berkeley Continuous Media Toolkit", in Multimedia Computing and Networking 1997, in Proc. SPIE 3020, pp 194-206, 1997.

[2] C. J. Lindblad, and D. L. Tennenhouse, "The VuSystem: A Programming System for Compute-Intensive Multimedia", in IEEE Journal of Selected Areas in Communications, 1996.

[3] K. Jonas, M. Kretschmer, and J. Modeker, "Get a KISS - Communication Infrastructure for Streaming Services in a Heterogeneous Environment", in Proc. of ACM Multimedia '98, Bristol, UK, pp. 401-410, 1998.

[4] S. Mungee, N. Surendran, and D. C. Schmidt, "The Design and Performance of a CORBA Audio/Video Streaming Service", in Proc. of the 32st Hawaii International Conference on System Systems(HICSS), Hawaii, January 1999.

[5] Object Management Group, The Common Object Request Broker: Architecture and Specification Revision 2.2, February 1998.

[6] D. Schmidt, D. Levine, and S. Mungee, "The Design and Performance of Real-time Object Request Brokers", Computer Communications, vol. 21, pp.294-324, April 1998.

[7] Object Management Group, Control and Management of A/V Streams specification, OMG Document telecom/97-05-07 ed., October 1997.

[8] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, RTP: A Transport Protocol for Real-Time Applications, IETF RFC 1889, January 1996.

[9] H. Schulzrinne, A. Rao, and R. Lanphier, Real-Time Streaming Protocol (RTSP), IETF RFC 2326, April 1998.

[10] C. Aurrecoechea, A. T. Campbell, and L. Hauw, "A Survey of QoS Architectures", ACM/Springer Verlag Multimedia Systems Journal , Special Issue on QoS Architecture, Vol. 6 No. 3, pp. 138-151, May 1998.

[11] S. N. Bhatti and G. Knight, "Enabling QoS adaptation decisions for Internet applications", Journal of Computer Networks, Vol. 31, No. 7, pp. 669-692, March 1999.

[12] Hyung-lll Kim and Sungyoung Lee, "Design of Media Object Architecture to Support Multimedia QoS", In Proc. of Korean Information Science Society 98, pp. 699-701, April 1998.

[13] J. Stankovic, S. Son and J. Liebeherr, "BeeHive: Global Multimedia Database Support for Dependable, Real-Time Applications", In Proc. of Second Workshop on Active Real-Time Databases, Lake Como, Italy, September 1997.