Agent-based Multimedia Data Sharing Platform

Anson Lee, Michael Lyu, Irwin King
Department of CSE
The Chinese University of Hong Kong
Shatin, New Territories, Hong Kong

Abstract *This paper proposes an agent-based platform which allows users to receive and exchange multimedia data from distributed sources, e.g. digital libraries, image databases and video databases. However, the multitude, diversity and the dynamic nature of multimedia data on the Internet make it difficult for us to access any specific piece of multimedia data. Nonetheless, we use software agents to (1) identify and collect audio and video files from heterogeneous information sources over the Internet, (2) accept queries from a client, (3) process and present the query results to the client and (4) index and maintain a centralized database for storing meta-data. Another important aspect of this platform is the transmission plan. It can help us to reserve enough bandwidth and to delivery multimedia data on the Internet.*

With the agent paradigm in our system, we are able (1) to increase system's robustness, (2) to increase system modularity and therefore make system maintenance easier, (3) to take advantage of distributed computing resources, (4) to make delegation of tasks easier in a centralized control, and (5) to make distributed control more feasible. In this paper, we describe the system architecture and the transmission plan for streaming audio or video files. Experimental results show that (1) our transmission plan can reserve enough bandwidth for audio and video delivery, (2) the platform can be extended easily to support a large number of concurrent client connections.

Keywords: software agent, digital library, distributed information retrieval, transmission plan

1 Introduction

Distributed file sharing system over the Internet has recently been a very hot topic in the academic as well as in the commercial and legal arena. The huge success and popularity of Napster's music sharing paradigm has taken the world by storm. Nonetheless, there are still interesting research issues surrounding the multimedia data sharing platform that require a more detailed examination.

The problem of distributed multimedia data retrieval can be solved by using software agents to identify the information sources. Hence, we designed and implemented an agent platform and a client application for multimedia data sharing. Our platform can support a number of information sources, e.g. digital libraries, data repositories, news sites, etc. The main features of our system are capable (1) to support the streaming MPEG audio and video over the Internet, (2) to make fast and accurate queries with the use of an user-friendly interface, (3) to scale up easily because the server is optimized for parallel processing, and (4) to allow efficient communications between agents as our inter-agent communication protocol is tailored for our agents which can minimize the overhead during communication.

There are other systems that are capable of supporting multimedia information access and retrieval [1, 2]. These systems use existing mobile agent platforms like Grasshopper [3] to develop a small-scale file sharing system which require users to install an agent execution environment on their computer. However, we do not require users to install any agent execution environment. Another distinguishing feature of our system is that we are capable to develop large-scale file sharing system. All meta data is stored in our server which make the workload of the client computer relatively low.

In the next section, we will include a survey of mobile-agent system, and briefly discuss their similarities and differences. Section 3 will introduce the system architecture and explain the use of each system component. Section 4 will focus on an audio transmission plan that we have designed and implemented. Section 5 will give some experimental results to demonstrate our system's performance characteristics under the multiple-client connection environment.

2 Survey of mobile agent systems

In this section, we will examine four representatives of mobile-agent system.

2.1 D'Agents

D'Agents, which was also known as Agent Tcl[4], support agents written in Tcl, Java and Scheme, as well as stationary agents written in C and C++. Like Ara, D'Agents provides a `go` instruction (Tcl and Java only), and automatically captures and restores the complete state of a migrating agent. Unlike Ara, only the D'Agents server is multi-threaded; each agent is executed in a separate process, which simplifies the implementation considerably, but adds the overhead of inter-process communication. The D'Agent server uses public-key cryptography to authenticate the identity of an incoming agent's owner. Stationary resource-manager agents assign access rights to the agent based on this authentication and the administrator's preferences, and language-specific enforcement modules enforce the access rights, either preventing a violation from occurring or terminating the agent when a violation occurs. Each resource manager is associated with a specific resource such as the file system. The resource managers can be as complex as desired, but the default managers simply associate a list of access rights with each owner. Unlike Ara, most resource managers are not consulted when the agent arrives, but instead only when the agent attempts to access the corresponding resource or explicitly requests a specific access right. At that point, the resource manager forwards all relevant access rights to the enforcement module, and D'Agents behaves in the same way as Ara, enforcing the access rights with short wrapper functions around the resource access functions.

D'Agents has been used in several information-retrieval applications, including the 3DBase, a system for retrieving three-dimensional drawings (CAD drawings) of mechanical parts based on their similarity to a query drawing.

2.2 Java-Based systems

Aglets. Aglets [5]was one of the first Java-based systems. Like all commercial systems, including Concordia, Jumping Beans and Voyager, Aglets does not capture an agent's thread state during migration, since thread capture requires modifications to the standard Java virtual machine. In other words, thread capture means that the system could be used only with one specific virtual machine, significantly reducing market acceptance. Thus, rather than providing the `go` primitive of D'Agents and Ara, Aglets and the other commercial systems instead use variants of the Tacoma model, where agent execution is restarted from a known entry point after each migration. In particular, Aglets uses an event-driven model. When an agent wants to migrate, it calls the `dispatch` method. The Aglets system calls the agent's `onDispatching` method, which performs application-specific cleanup, kills the agent's threads, serializes the agent's code and object state, and sends the code and object state to the new machine. On the new machine, the system calls the agent's `onArrival` method, which performs application-specific initialization, and then calls the agent's `run` method to restart agent execution.

Aglets includes a simple persistence facility, which allows an agent to write its code and object state to secondary storage and temporarily "deactivate" itself; proxies, which act as representatives for Aglets, and among other things, provide location transparency; lookup service for finding moving Aglets; and a range of message-passing facilities for inter-agent communication. The Aglet security model is similar to both the D'Agent and Ara security models, and to the security models for the other Java-based systems below. An Aglet has both an owner and a manufacturer. When the agent enters a context (i.e., a virtual place) on a particular machine, the context assigns a set of permissions to the agent based on its authenticated owner and manufacturer. These permissions are enforced with standard Java security mechanisms, such as a customized security manager.

Concordia. Concordia is a Java-based mobile agent system that has a strong focus on security and reliability. Like most other mobile-Java agent systems, they move the agent objects code and data, but not thread state, from one machine to another. Like many other systems, Concordia agents are bundled with an *itinerary* of places to visit, which can be adjusted by the agent. If the remote site is not currently reachable, agents, events and messages can be queued. Agents are carefully saved to a persistent store, before departing a site and after arriving at a new site, to avoid agent loss in the event of a machine crash. Agents are protected from tampering through encryption while they are in transmission or stored on disk; agent hosts are protected from malicious agents through cryptographic authentication of the agent's owner, and access control lists that guard each resource.

Jumping Beans. Jumping Beans is a Java-based framework for mobile agents. Computers wishing to host mobile agents run a Jumping Beans *agency*, which is associated with some Jumping Beans *domain*. Each domain has a central server, which authenticates the agencies joining the domain. Mobile agents move from agency to agency, and agents can send messages to other agents; both mechanisms are implemented by passing through the server. Thus the server becomes a central point for tracking, managing, and authenticating agents. It also becomes a central point of failure or a performance bottleneck, although they intend to develop scalable servers to run on parallel machines. Another approach to scalability is to create many small domains, each with its own server. In the current version, agents cannot migrate between domains, but they intend to support that capability in future versions. Security and reliability appear to be important concerns of their system; public-key cryptography is used to authenticate agencies to the server, and vice versa; access-control lists are used to control an agent's access to resources, based on the permissions given to the agent's owning user.

Although they claim to move all agent code, data, and state, it is not clear from their documentation whether they actually move thread state, as in Agent Java. They require that the agent be a serializable object, so it seems likely that they implement the weaker form of mobility common to other Java-based agent systems.

2.3 Similarities and differences

All mobile-agent systems have the same general architecture: a server on each machine accepts incoming agents, and for each agent, starts up an appropriate execution environment, loads the agent's state information into the environment, and resumes agent execution. Some systems, such as the Java-only systems above, have multi-threaded servers and run each agent in a *thread* of the server process itself; other systems have multi-process servers and run each agent in a separate interpreter process; and the rest use some combination of these two extremes. D'Agents, for example, has a multi-threaded server to increase efficiency, but separate interpreter processes to simplify its implementation. Jumping Beans is of particular note since it uses a centralized server architecture (in which agents must pass through a central server on their way from one machine to another). rather than a peer-to-peer server architecture (in which agents move directly from one machine to another). Although this centralized server easily can become a performance bottleneck, it greatly simplifies, tracking, administration and other issues, perhaps increasing initial market acceptance.

Currently, for reasons of portability and security, nearly all mobile-agent systems either interpret their languages directly, or compile their languages into bytecodes and then interpret the bytecodes. Java, which is compiled into bytecodes for the Java virtual machine, is the most popular agent language, since (1) it is portable but reasonably efficient, (2) its existing security mechanisms allow the safe execution of untrusted code, and (3) it enjoys widespread market penetration. Java is used in all commercial systems and in several research systems. Due to the recognition that agents must execute at near-native speed to be competitive with traditional techniques in certain applications, however, several researchers are experimenting with "on-the-fly" compilation. The agent initially is compiled into bytecodes, but compiled into native code on each machine that it visits, either as soon as it arrives or while it is executing. The most recent Java virtual machines use on-the-fly compilation, and the Java-only mobile-agent systems, which are not tied to a specific virtual machine, can take immediate advantages of the execution speedup.

Mobile-agent systems generally provide one of two kinds of migration: (1) `go`, which captures an agent's object state, code, and control state, allowing it to continue execution from the exact point at which it left off; and (2) *entry point*, which captures only the agent's object state and code, and then calls a known entry point inside its code to restart the agent on the new machine. The `go` model is more convenient for the end programmer, but more work for the system developer since routines to capture control state must be added to existing interpreters. All commercial Java-based systems use entry-point migration, since market concerns demand that these systems run on top of unmodified Java virtual machines. Research systems use both migration techniques.

Finally, existing mobile-agent systems focus on protecting an *individual* machine against malicious agents. Aside from encrypting an agent in transit and allowing an agent to authenticate the destination machine before migrating, most existing systems do not provide any protection for the agent or for a group of machines that is not under single administrative control.

Other differences exist among the mobile-agent systems, such as the granularity of their communication mechanisms, whether they are built on top of or can interact with CORBA, and whether they conform to the emerging mobile-agent standards. Despite these differences, however, all of the systems discussed above (with the exception of Messengers, which is a lighter-weight mobile-agent system) are intended for the same applications, such as workflow, network management, and automated software installation. All of the systems are suitable for distributed information retrieval, and the decision of which one to use must be based on the desired implementation language, the needed level of security, and the needed performance.

3 System architecture

Our platform is a multi-agent system in which three semi-autonomous agents interact or work together to perform a user's goal. They are the *Server Agent(SA)*, *Query Agent(QA)* and *Database Agent(DA)*. Figure 1 shows the complete system architecture and the relationships between the platform and client computers. Server side agents such as the SA, the QA and the DA are running in Agent Tcl [4] which is built on top of the server machine. In addition, a database is installed for storing meta-data. This design allows us to develop a thin client application. The advantage of using thin client

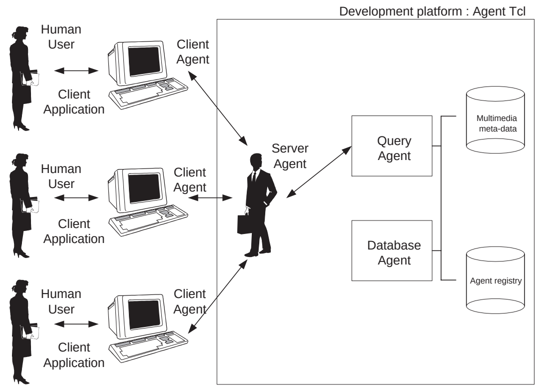

Figure 1: The system architecture.

application is to reduce the CPU consumption in user's computer.

In the client application, there is a *Client Agent(CA)* for communicating with the SA. User request will be received from the CA's user interface and forwarded it to SA. The CA is also responsible for presenting the query results to the user.

3.1 System Components

MySQL. MySQL is a database management system. A database is a structured collection of data. We stores data in separate tables rather than putting all the data in one big storeroom. This adds speed and flexibility. The tables are linked by defined relations making it possible to combine data from several tables on request. The reason of choosing MySQL is that MySQL is very fast, reliable, and easy to use. MySQL also has a very practical set of features developed in very close cooperation with our users. The most important, it is open source and free of charge. Open source means that it is possible for anyone to use and modify. Anybody can download MySQL from the Internet and use it without paying anything. Anybody so inclined can study the source code and change it to fit their needs.

Agent Tcl. As we have mentioned before, Agent Tcl is an agent development platform. The reasons of using it are (1) easy to scale up, (2) allow an agent to choose the best migration strategy given the current network and (3) support for mobile-computing environments, where applications must deal with low-bandwidth, high-latency and unreliable network links.

3.2 Roles of the Agents

An user initiates the query for multimedia data by issuing a request to the CA. The CA requests for multimedia data specifications from the user. The specification must include a description of the multimedia content. For audio file like songs, it should include the name of the song, the singer's name, and the music category. For video file like news, it should include the news category, the reporter's name and the date of that news. At the same time, the CA will register itself to the agent registry database and submit a list of multimedia data available in the user's computer to the SA. The SA is responsible for dispatching client's requests to the QA or the DA. If the client's request is a query, it will be forwarded to the QA. The QA is responsible for making query to find out the site containing the requested multimedia data. If the client's request is a list of multimedia data, it will be forwarded to the DA for indexing and updating the multimedia meta-data database. Moreover, the DA will be asked to update the database if a multimedia data is removed from a particular site.

3.3 Client Application

The client application supports agents with the following missions: (1) search MPEG audio files and MPEG video files by user's specification, (2) stream and play audio files and video files on a remote location. It can decode two types of multimedia file formats including MPEG I layer III audio and MPEG I video. However, we face problems in efficiently transfer audio and video files over the Internet. The bursty, variable-bit-rate traffic complicates the effort to allocate network resources to ensure continuous playback at client sites. Our approach for smoothing compressed video or au-

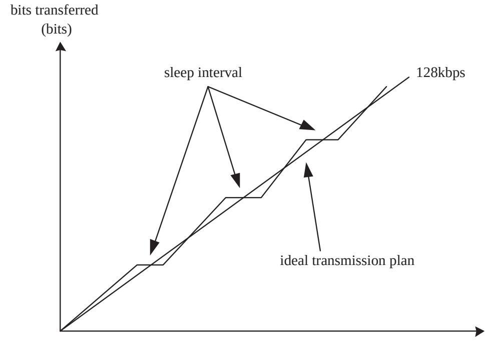

Figure 2: Transmission plan.

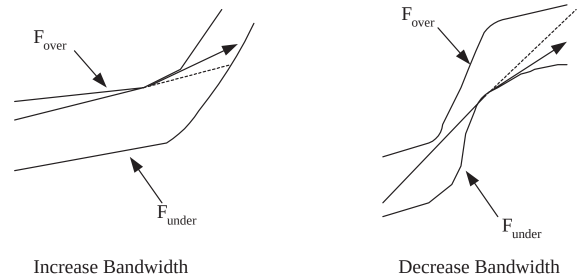

Figure 3: Valid transmission plan.

dio file is try to achieve a constant transmission rate for the entire delivery sequence.

4 Transmission Plan

The transfer of multimedia data on the Internet requires server to support large fluctuations in bandwidth requirements. The bursty, variable-bit-rate traffic complicates the effort to allocate server and network resources to ensure continuous playback at client sites. Some of the famous transmission plans are Critical Bandwidth Allocation, Minimum Changes Bandwidth Allocation, Minimum Variability Bandwidth Allocation and Piecewise Constant Rate Transmission and Transport Algorithm. All of them require an intensive CPU consumption and they have different performance towards the peak rate requirements, the number of bandwidth changes, the variability of the bandwidth allocations and the variability of the time between bandwidth changes. However, the distinguishing feature of our plan is that we are capable to reserve enough bandwidth for the transmission but still have a low CPU consumption.

For the audio delivery, we try to use the following transmission plan to approximate the bit rate requirement of streaming MPEG audio which is 128kbps [6][7].

The bit rate requirement of streaming MPEG audio can be approximated by adding some sleeping intervals before we transmit a new audio packet. The use of this transmission plan is to ensure the sender must always transmit quickly enough to avoid buffer underflow in the client side. Similarly, it should not allow a client to receive too much data to prevent overflow of the playback buffer.

Suppose time slot i requires f_i bytes of storage, the lower bound and the upper bound of the transmission rate are approximated by:

$$F_{under}(k) = \sum_{i=0}^{k} f_i$$

and

$$F_{over}(k) = b + \sum_{i=0}^{k} f_i$$

where $k = 0, 1, ..., n-1$ and b is the size of the playback buffer.

Experimental results show that our transmission plan is valid because it satisfies the following inequality:

$$F_{under}(k) \leq \sum_{i=0}^{k} c_i \leq F_{over}(k)$$

where c_i is the transmission rate during time slot i of the smoothed audio stream. The above inequality is shown in Figure 3.

5 Experimental Results

Given a query to search for an audio file by keyword matching, we are able to measure the query time and time delay in delivering the file to the user. This experiment was done on a PC with a 300MHz PII Intel CPU, 128Mb memory and a 12Gb harddisk. Our platform was built on top of Redhat Linux 6.2 with kernel-2.2.16. In this experiment, we want to show that our system can support a large number of concurrent connections without too much performance degradation. The result is summarized in Table 1. In this experiment, we try

Table 1: Measurement of Average Query Time

Number of concurrent connection	Average query time
1	0.76sec
10	0.96sec
20	1.07sec
40	1.12sec

to run all client applications on a single computer. We find that the query time is affected by number of concurrent connections. They have a linear relationship. We believe that if we run client applications in separate computers, the query time can be shorten even further. However, the delay is still acceptable in this experiment. It can be shown that the use of agent paradigm allows us to scale up the system easily, increase the its robustness, and to minimize the query time.

6 Conclusion

In this paper, we have introduced our agent platform for sharing multimedia data. We discussed the system architecture and the role of agents. Moreover, we implemented a simple transmission plan to deliver audio or video files over the Internet. Experimental results show that the transmission plan not only utilizes the bandwidth efficiently but also preserves the quality of the multimedia data content. Moreover, results show that our system can be scaled up easily but still keep a short query time.

7 Acknowledgments

The work described in this paper was fully supported by the Research Grants Council of the Hong Kong Special Administrative Region, with Project Numbers CUHK4193/00 and CUHK4407/99E.

References

[1] A. Karmouch F. Ziade. A multimedia transportable agent system. In *Proc. of IEEE Canadian Conf. Elec. and Computer Engineering*, 1997.

[2] B. Falchuk and A. Karmouch. A mobile agent prototype for autonomous multimedia information access, interaction and retrieval. In *Proceedings of Multimedia Modeling*, 1997.

[3] M. Breugst, I. Busse, S. Covaci, and T. Magedanz. Grasshopper - a mobile agent platform for in based service environments. In *IEEE IN Workshop 1998*, May 1998.

[4] Gray R.S. Agent tcl: A flexible and secure mobile-agent system. In *Proc. of the Fourth Annual Tcl/Tk Workshop, Monteray CA*, 1996.

[5] Lange D. and Chang D.T. Aglets workbench - programming mobile agents in java. *White Paper, IBM Corporation, Japan*, 1996.

[6] Peter Noll. Mpeg digital audio coding. *IEEE Signal Processing Magazine*, pages 59–81, 1997.

[7] Davis Pan. A tutorial on mpeg/audio compression. *IEEE Multimedia*, pages 60–74, 1995.