Design and Implementation of a QoS-aware Replication Mechanism for a Distributed Multimedia System

Giwon On[1], Jens B. Schmitt[1] and Ralf Steinmetz[1,2]

1
Industrial Process and System Communications
Dept. of Electrical Eng. &
Information Technology
Darmstadt University of Technology
Merckstr. 25 • D-64283 Darmstadt • Germany

2
GMD IPSI
German National Research Center
for Information Technology
Dolivostr. 15 • D-64293 Darmstadt • Germany

Mail: {Giwon.On, Jens.Schmitt, Ralf.Steinmetz}@KOM.tu-darmstadt.de

Abstract. This paper presents the design and implementation architecture of a replication mechanism for a distributed multimedia system *medianode* which is currently developed as an infrastructure to share multimedia-enhanced teaching materials among lecture groups. The proposed replication mechanism supports the quality of service (QoS) characteristics of multimedia data and the availability of system resources. Each type of data handled and replicated are classified according to their QoS characteristics and replication requirements. The main contribution of this paper is the identification of new replication requirements in distributed multimedia systems and a multicast-based update propagation mechanism by which not only the update events are signalled, but also the updated data are exchanged between replication managers. By prototyping the proposed replication mechanism in medianode, we prove the feasibility of our concept for combining the QoS concept with replication mechanisms.

1 Introduction

One major problem about using multimedia material in lecturing is the trade-off between actuality of the content and quality of the presentations. A frequent need for content refreshment exists, but high quality presentation can not be authored by the individual teacher alone at the required rate. Thus, it is desirable that teachers presenting the same or at least similar topics but work at different locations can easily share their multimedia-enhanced lecture materials. The medianode project[1] is intended to provide such means for sharing to lecturers at the universities of the German state Hessen.

The design of the medianode system addresses issues of availability, quality of service, access control and distribution of data. To support teachers, it must allow for transparent access to shared content, and it must be able to operate in disconnected mode since lecturers do not have access to the network at all times during their presentations. The medianode system architecture is intended for de-centralized operation of a widely distributed system. Within this distributed system, each participating host is called a medianode and conceptually equal to all other participating nodes, i.e. a medianode is not considered a client or a server. Client or server tasks are taken on by medianodes in the system depending on the their resources and software modules.

Replication is the maintenance of on-line copies of data and other resources[2, 5]. Replication of presentation materials and meta-data is an important key to providing high availability, fault tolerance and quality of service (QoS) in distributed multimedia systems[21], and in particular in medianode. For example, when a user requires access (read/write) to a presentation which comprises audio/video data and some resources which are not available in the local medianode at this point of time, a local replication manager copies the required data from their original location and puts it into either one of the medianodes located nearby or the local medianode without requiring any user interaction (user transparent). This function enhances the total performance of medianode by reducing the response delay that is often caused due to insufficient system resources, such as memory, CPU time, and network bandwidth, at a given service time. Furthermore, because of the available replica in the local medianode, the assurance that users can continue their presentation in a situation of network disconnection, is significantly higher than without replica.

The structure of the paper is as follows. In Section 2, we give an overview of related work. The merits and limitations of existing replication mechanisms are discussed and a comparison of our approach with previous work is given. Section 3 presents our replication system model. We define the scope of our replication mechanism in medianode and present the characteristics of presentational media types, for which we identify a need for new replica units and granularity. We also describe the proposed replication maintenance concept, e.g. how and when replicas are created and how the updates are signalled and transported. In Section 4, we present our prototype implementation architecture. It describes operation flows in medianode with and without replication system. We conclude the paper with a summary of our work and an outlook towards possible future extensions of our replication mechanism in Section 5.

2 Related Works

Several approaches to replication have already been proposed. The approaches differ for distributed file systems from those for Internet-based distributed web servers and those for transaction-based distributed database systems. Well-known replication systems in distributed file systems are AFS[6], Coda[7], Rumor[13], Roam[14] and Ficus[16] which keep the file service semantics of Unix. Therefore, they support to develop applications based on them. They are based either on a client-server model or a peer-to-peer model. Often they use optimistic replication which can hide the effects of network latencies. Their replication granularity is mostly the file system volume, with a large size and low number of replicas. There is some work on optimization for these examples concerning of update protocol and replica unit. To keep the delay small and therefore maintain real-time interaction, it was desirable to use an unreliable transport protocol such as UDP. In the earlier phases, many approaches used unicast-based data exchange, by which the replication managers communicated with each other one-to-one. This caused large delays and prevented real-time interaction. To overcome this problem, multicast-based communication has used recently [9, 11, 12, 17]. For Coda, the RPC2 protocol is used for multicast-based update exchange, which provides with the *Side Effect Descriptor* transmission of large files.

For limiting the amount of storage used by a particular replica, Rumor and Roam developed the selective replication scheme[18]. A particular user, who only needs a few of the files in a volume, can control which files to store in his local replica with selective replication. A disadvantage of selective replication is the ‘full backstoring’ mechanism: if a particular replica stores a particular file in a volume, all directories in the path of that file in the replicated volume must also be stored.

JetFile[9] is a prototyped distributed file system which uses multicast communication and optimistic strategies for synchronization and distribution. The main merit of JetFile is its multicast-based callback mechanism by which the components of JetFile, such as file manager and versioning manager interact to exchange update information. Using the multicast-based callback, JetFile distributes the centralized update information which is normally kept by the server over a number of multicast routers. However, the multicast callbacks in JetFile are not guaranteed to actually reach all replication peers, and the centralized versioning server, which is responsible for serialization of all updates, can lead to a overloaded system state. Furthermore, none of the existing replication systems supports quality of service (QoS) characteristics of (file) data which they handle and replicate.

3 Replication System Model

3.1 Design Goals and Scope of our Replication System

By analysing the service requirements distributed multimedia systems for the example of medianode, we identified a number of issues that the design of our replication system needs to address:

- **High availability:** The replication system in medianode should enable data/service access in both connected and disconnected operation modes. Users can keep multiple copies of their files on different medianodes that are distributed geographically across several universities in the state of Hessen.
- **Consistency:** Concurrent updates and system failures can lead to replicas not being consistent any more, i.e. *stale* state. The replication system should offer mechanisms for both resolving conflicts and keeping consistency between multiple replicas and their updates.
- **Location and access transparency:** Users do not need to know where presentation resources are physically located and how these resources are accessed.
- **Cost efficient update transport:** Due to the limitation of system and network resources, the replication system should use multicast-based transport mechanism for exchanging updates to reduce resource utilization.
- **QoS support:** The specific characteristics of presentational data, especially of multimedia data should be supported by the proposed replication mechanism.

In medianode, we mainly focus on the replication service for accessing data in terms of ‘*inter-medianode*’, i.e. between medianodes, by providing replica maintenance in each medianode. Consequently, a replication manager can be implemented as one or a set of medianode’s bow instances in each medianode. The replication managers communicate among each other to exchange update information through the whole medianodes.

3.2 Different Types of Presentation Data

Data organization comprises the storage of content data as well as meta information about this content data in a structured way. The typical data types which can be identified in medianode are described in [3]. Table 1 shows an overview of these data types with their characteristics.

Table 1: Data categories and their characteristics in medianode

target data	availability req.	consistency req.	persistency	update frequency	data size	QoS playback
presentation description	high	middle (high)	yes	low	small/ middle	not required
organizational data	high	high	yes	low	small	not required
file/data description	high	middle	yes	middle	small	not required
multimedia resources	high	middle	yes	middle	large	required
system resources	middle (low)	middle	no	high	small	not required
user session/ token	high	high	no	high	small	not required

3.3 Classification of Target Replicas

As argued in subsection 3-1, the main goal of replication is to provide availability of medianode's services and to decrease the response time for accesses to data located on other medianodes. To meet this goal, data which is characterized by a high availability requirement (See Table 1) should be replicated among the running medianodes. We classify different types of target replicas according to their granularity (data size), requirement of QoS support, update frequency and whether their data type is 'persistent' or not ('volatile'). Indeed, there are three classes of replicas in medianode:

- *Truereplicas* which are persistent and of large size. Content files of any media type, which also may be parts of presentation files are Truereplicas. Truereplicas are the only replica type from the three types, to which the end users have access for direct manipulation (updating). On the other side, these are also the only replica type which requires the support of really high availability and QoS provision.
- *Metareplicas* (replicated metadata objects) that are persistent and of small size. An example would be a list medianodes (sites) which currently contain an up-to-date copy of a certain file. This list itself is replicated to increase its availability and improve performance. A metareplica is a replica of this list.

- *Softreplicas* which are non-persistent and of small size. This kind of replicas can be used for reducing the number of messages exchanged between the local and remote medianodes, and thereby reducing the total service response time. I.e., if a local medianode knows about the available local system resources, then the local replication manager can copy the desired data into the local storage bow, and the service that is requested from users which requires exactly the data can be processed in a shorter response time. Information about the available system resource, user session and the validity of user tokens are replicas of this type.

All replicas which are created and maintained by our replication system are an identical copy of original media. Replicas with errors (non-identical copy) are not allowed to be created. Furthermore, we also do not support any replication service for function calls, and elementary data types.

3.4 The Replication Mechanism

3.4.1 Replication Model

Basically, our replication system does not assume a client-server replication model, because there are no fixed clients and servers in the medianode architecture; every medianode may be client or server depending on its current operations. Peer-to-peer model with the following features is used for our replication system:

(a) Every replica manager keeps track of a local file table including replica information.

(b) Information whether and how many replicas are created is contained in the every file table. I.e. each local replica manager keeps track of which remote replica managers (medianode) are caching which replicas.

(c) Any access to the local replica for reading is allowed, and guaranteed that the local cached replica is valid until notified otherwise.

(d) If any update happens, the corresponding replica manager sends a multicast-based update signal to the replica managers which have the replica of the updated replica and therefore members of the multicast group.

(e) To prevent excessive usage of multicast addresses, the multicast IP addresses through which the replica managers communicate can be organized in small replica sub-groups. Examples for such sub-groups are file directories or a set of presentations about a same lecture topic.

3.4.2 Update Distribution and Transport Mechanism

The update distribution mechanisms in medianode differs between the three replica types and their concerning managers. This is due to the fact that the three replica types have different levels of requirements on and characteristics of high availability, update frequency and consistency QoS (see Table 1). Experience from GLOVE[4] and [5] also shows that differentiating update distribution strategies makes sense for web and other distributed documents. The medianode's replication system offers an unique interface to the individual update signalling and transport protocols which are selectively and dynamically loaded and unloaded from the replica transport manager that is implemented as an instance of medianode's access bow. Possible update transport and signalling protocols are:

- RPC protocol [2,10]as a simple update distribution protocol. This mechanism is mainly used at the first step of our simple and fast implementation.
- A *multicast based RPC* communication mechanism. In this case, the updates are propagated via multicast other replica managers which are members of the multicast group. *RPC2* [7,11] is a good candidate for the first implementation. *RPC2* also offers the transmission of large files, such as the updated AV content files or *diff*-files, by using the *Side Effect Descriptor.* But, the *RPC2* with *Side Effect Descriptor* does not guarantee any reliable transport of updates.
- LC-RTP based reliable multicast protocol: LC-RTP (Loss Collection-Realtime Transport Protocol)[12] is originally developed as an extension of RTP protocol to support the reliable video streaming within the medianode project. Therefore, we adopt LC-RTP and check the usability of the protocol, depending on the degree of reliability required for the individual groups of replicas.

3.4.3 Approaches for Solving Conflicting Updates and for Resolving Conflicts

The possible conflicts that could appear during the shared use of presentational data and files are either (a) update conflict when two or more replicas of an existing file are concurrently updated, (b) naming conflict when two (or more) different files are given concurrently the same name, and (c) update/delete conflict that occur when one replica of a file is updated while another is deleted. In most existing replication systems, the conflict resolving problem for update conflicts was treated as a minor problem. It was argued that most files do not get any conflicting updates, with the reason that only one person tends to update them[9]. Depending on the used replication model and policy, there are different approaches of which our replication system uses the following strategies [7,8,13,14,15]:

- Swapping - to exchange the local peer's update with other peer's updates;
- Dominating - to ignore the updates of other peers and to keep the local update as a final update;
- Merging - to integrate two or more updates and build one new update table;

4 Implementation

We have implemented a prototype of the proposed replication system model on a Linux platform (SuSe 7.0, Redhat 6.2). Implemented are the media (file) and its replica manager (ReplVerifierBow), update transport manager (ReplTransportBow), replica service APIs which are Unix-like file operation functions such as open, create, read, write, close (ReplFileSysBow), and a session information storage bow (VolatileStorBow) which maintains user's session and token information.

4.1 Bow Description

Table 2 gives a short descriptions of bows which implement medianode's basic functions and the services of our replication system.

Table 2: A summary of medianode's bows used for our replication system

Bow	Description
MNBow	• addressible via an unique bow identifier and version number • uses request and response queues and dispatcher threads • defines request processing routine
MNAccessBow	• child class of MNBow and implements access bow API • offers RPC server modules and enables RPC connection from web server • HTML-based presentation files are provided via this bow
ReplTransportBow	• child class of MNBow and a variant of access bow • implements the transport managers for replication service • offers transport protocol modules such as RPC, RPC2, and LC-RTP (LC-FTP)
(GUI)TelnetBow	• child class of MNBow and a variant of access bow • offers TCP server modules • acts as telnet server and provides information which medianode maintains such a list of bows loaded by the core, memory usages of a certain medianode's process running etc. • GUIBow implements a TelnetBBow with a graphical user interface
MNVerifierBow	• child class of MNBow and implements verifier bow API • offers modules needed for user authentication
ReplVerifierBow	• child class of MNBow and a variant of verifier bow • implements the media (file) and replica managers for replication service • maintains the three replica tables
MNStorageBow	• child class of MNBow and implements storage bow API
FileBow	• child class of MNStorageBow • implements functions for local file operation • no interface routine support for replication service
XMLBow	• child class of MNStorageBow • offers modules for dynamic generation of presentation files • no interface routine support for replication service
ReplFileSysBow	• child class of MNStorageBow • implements functions for local file operation • implements interface routines for replication service
VolatileBow	• child class of MNStorageBow • implements functions for maintaining volatile data such as memory usage information • implements interface routines for replication service

4.2 Presentation Service without Replication Support

The interaction model for medianode's bows is based on a 'request-response' communication mechanism. A bow which needs to access data or services creates a request packet and sends it to the core. According to the request type, the core either processes the request packet directly, or forwards it to a respective bow. The processing results are sent to the origin bow in a response packet. The request and response packets contain all necessary information for the communication between bows as well as for processing the requests.

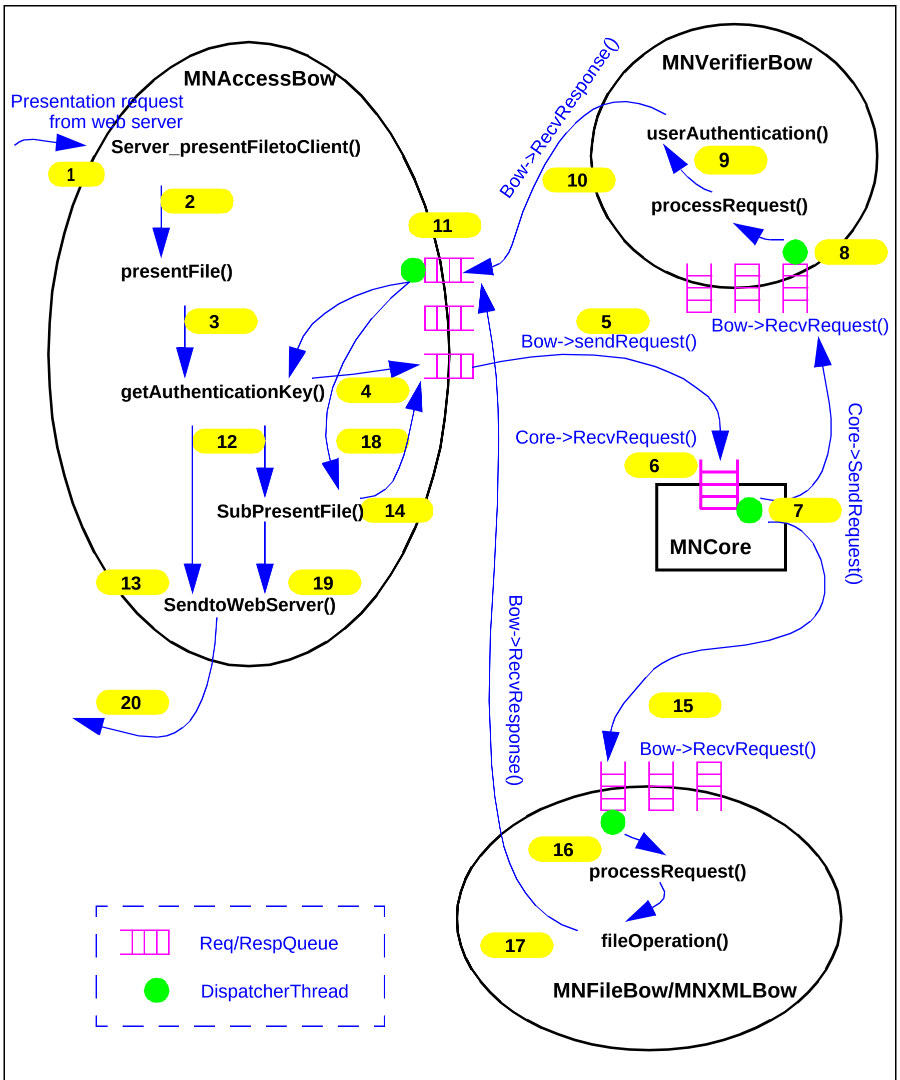

Figure 1: medianode: internal service flow without a replication support

Based on this request-response mechanism, we experimented some presentation scenarios with and without a replication service. Figure 1 shows one example of presentation services without a replication system. Upon receiving a presentation request from user via web server, the MNAccessBow creates first a request packet to check user's authentication (steps 1~4) and sends it via the core (steps 5~7) to MNVerifierBow which puts authentication test value into a response packet and sends it to the origin bow, MNAccessBow (steps 8~11). In the case of a successful authentication, MNAccessBow creates a request packet to get the required presentation data and sends it via the core to a corresponding storage bow (steps 12~15). Either MNFileBow or MNXMLBow, it depends on the requested (media) data type, checks whether the data exists locally, and then creates a response packet which contains either a file handle or an error message, sends it to the MNAccessBow (steps 16~18). MNAccessBow sends then to the web server a response which is either an authentication failure message or a presentation file.

4.3 Presentation Service with Replication Support

4.3.1 Initialization of MediaList and Replica Tables

In this subsection, we describe the medianode's operation flow with the replication service. Basically, the replication service in medianode begins by creating media list and replica tables of the three replica types in each medianode. As shown in Figure 2, ReplFileSysBow sends a request packet via the core to ReplVerifierBow for creating a media list for media data which locate in the local medianode's file system (steps 1~2). Upon receiving the request packet, ReplVerifierBow creates media list which will be used to check the local availability of any required media data (step 3). ReplVerifierBow then builds the local replica tables for the two replica types, *Truereplicas* and *Metareplicas*, if the replica information exists already. A medianode configuration file can specify the default location where replica information is stored. Every type of replica table contains a list of replicas with the information about organization, replica volume identifier, unique file name, file state, version number, number of replicas, a list of replica, a multicast IP address, and some additional file attributes, such as access right, creation/modification time, size, owner, and file type. The third replica table for the *Softreplicas* to which the local system resource, user session and token information belong may be needed to be created in terms of memory allocation, and the contents of this table can be partly filled when users request some certain services. Once the replica tables are created, they are stored in the local file system and accessible persistently.

4.3.2 Maintaining Replica Tables

In medianode, these three replica tables are maintained locally by the local replication manager. So, there is no need to exchange any update-related messages for the files of which there is no replica created. This approach increases the system resource utilization, especially network resources, by decreasing the message numbers exchanged between the replication managers among the distributed medianodes. But, when any medianode wants to get a replica from the local replica tables, the desired replica elements are copied to the target medianode, and the replication manager at the target medianode keeps these replica elements separate in another replica table which is used only for the management of remote replicas, i.e. for the management of replicas for which their original files are stored in a remote medianode.

4.3.3 Acquiring a Replica to Remote Replication Managers

Upon receiving the service requests (data access request) from users, the local medianode attempts to access the required data in a local storage bow (ReplFileSysBow) (step 4~5).

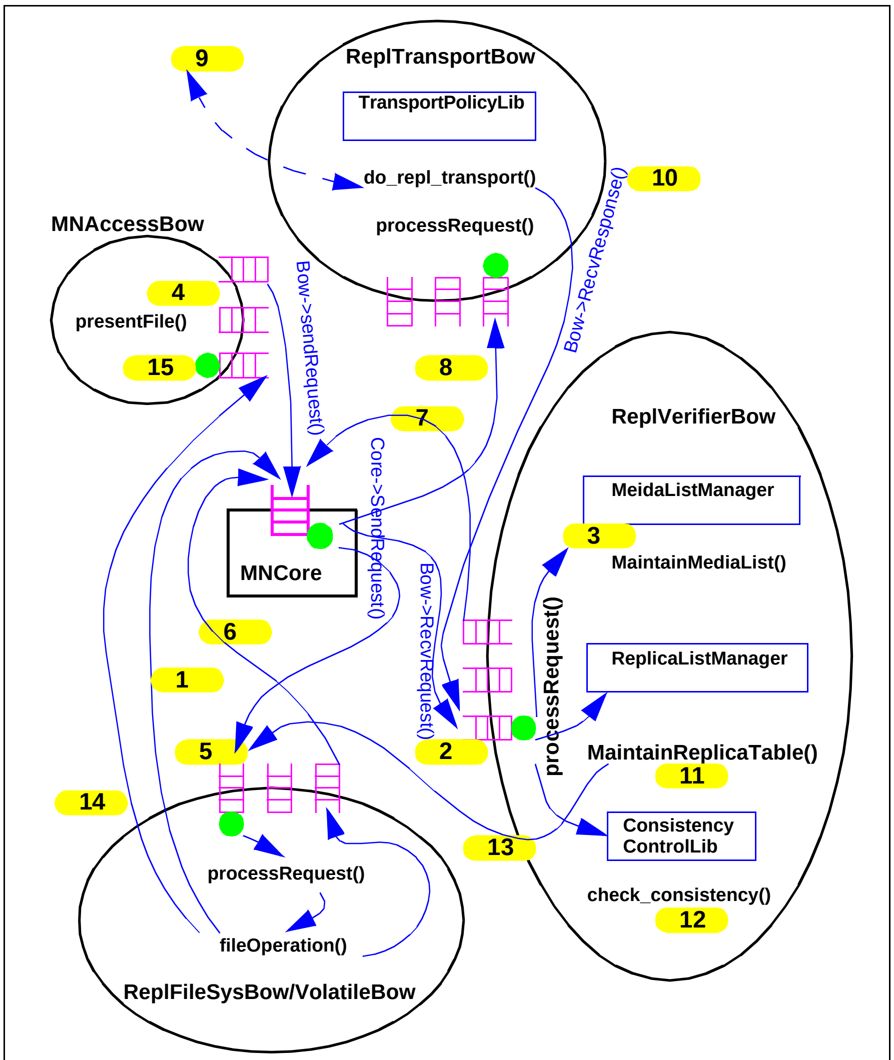

Figure 2: medianode: internal service flow with a replication support

In the case, when the data is not available locally, the local ReplFileSysBow sends a request packet to ReplVerifierBow to get a replica for the data. The ReplVerifierBow then start a process to acquire a replica by creating a corresponding request packet which is passed to ReplTransportBow (steps 6~8). The ReplTransportBow multicasts a data search request to all the peer replication managers and waits for replication managers to respond (step 9). The list of medianodes to which the multicast message is sent can be read from the medianode's configuration file. Whether the ReplTransportBow waits for all responses or receives the first one is dependent on the optimization policy which is given as configuration flag. After receiving the target replica, the ReplTransportBow sends a response packet to the ReplVerifierBow which then updates the corresponding replica tables, i.e. ReplVerifierBow adds the new replica element to the Truereplicas table and its metadata to the Metareplicas table, respectively (steps 10~13). Finally, the local ReplFileSysBow which originally issued replica creation request creates a response packet including the replica handle and then sends it to the MNAccessBow (steps 14~15).

5 Summary and Future Works

In this paper, we presented the replication mechanism of our distributed media system *medianode*, and described the design and implementation architecture of the prototyped replication system. The main contributions of this paper are (1) to identify the new replication requirements for distributed multimedia systems, and (2) to build a replication mechanism for distributed multimedia systems, which supports the individual QoS characteristics of multimedia data and uses system resource usage information. To achieve these targets, we first studied the characteristics of presentational media types which are handled in medianode, identified replica units and granularity. These have not been considered and not supported in existing replication mechanisms. We then built a QoS-aware replication mechanism, in which the decision whether and when a replica should be created from original file is made by checking the QoS characteristics of the requested media and the current usages of available system resources, for distributed multimedia systems based on the new requirements and the result of feature surveys.

The next working steps would be to design other replication services which provide service implementations such as predictive replication to increase access availability and to reduce latency. Similar approaches are Hoarding[7, 19], prefetched caching and resource reservation in advance[20].

6 References

[1] The medianode project. (web site http://www.httc.de/medianode).

[2] G. Coulouris, J. Dollimore and T. Kindberg. *Distributed Systems*, 3rd Ed., Chapter 1,8,14 and 15, Addison-Wesley, 2001.

[3] G. On, M. Zink, M. Liepert, C. Griwodz, J. Schmitt, R. Steinmetz. Replication for a Distributed Multimedia System. In *Proc. of ICPADS2001*, pp.37-42, 2001.

[4] G. Pierre, I. Kuz, M. van Steen and A.S. Tanenbaum. Differentiated Strategies for Replicating Web documents, In *Proc. of 5th International Workshop on Web*

Caching and Content Delivery, Lisbon, May 2000.

[5] P. Triantafillou and D.J. Taylor. Multiclass Replicated Data Management: Exploiting Replication to Improve Effciency. In *IEEE Trans. on Parallel and Distributed Systems*, pages 121-138, Vol.5, No.2, Feb.1994.

[6] R. Campbell. *Managing Andrew File System (AFS*), Prentice Hall PTR, 1998.

[7] M. Satyanarayanan, J.J. Kistler, P. Kumar, M.E. Okasaki, E.H. Siegel, and D.C. Steer. Coda: A Highly Available File System for a Distributed Workstation Environment. In *IEEE Transaction on Computers*, 39(4), April 1990.

[8] J. Yin, L. Alvisi and C. Lin. Volume Leases for Consistency in Large-Scale Systems. In *IEEE Trans. on Knowledge and Data Engineering*, 11(4), 1999.

[9] B. Groenvall, A. Westerlund and S. Pink. The Design of a Multicast-based Distributed File System. In *Proceedings of Third Symposium on Operating Systems Design and Implementation, (OSDI'99), New Orleans, Louisiana,* pages 251-264. February, 1999.

[10] K.P. Birman and B.B. Glade. Reliability Through Consistency. In *IEEE Software*, pages 29-41, May 1995.

[11] M. Satyanarayanan and E.H. Siegel. Parallel Communication in a Large Distributed Environment. In *IEEE Trans. on Computers*, pages 328-348, Vol.39, No.3, March 1990.

[12] M. Zink, A. Jones, C. Girwodz and R. Steinmetz. LC-RTP (Loss Collection RTP): Reliability for Video Caching in the Internet. In *Proc. of ICPADS 2001: Workshop*, pages 281-286. IEEE, July 2000.

[13] R. Guy, P. Reiher, D. Ratner, M. Gunter, W. Ma, and G. Popek. Rumor: Mobile Data Access Through Optimistic Peer-to-Peer Replication. In *Workshop on Mobile Data Access,* November 1998.

[14] D. Ratner, P. Reiher, and G. Popek. Roam: A Scalable Replication System for Mobile Computing. In *Workshop on Mobile Databases and Distributed Systems (MDDS),* September 1999.

[15] P. Triantafillou and C. Neilson. Achieving Strong Consistency in a Distributed File System. In *IEEE Transaction on Software Engineering*, pages 35-55, Vol.23, No.1, January 1997.

[16] T.W. Page,Jr., R.G. Guy, G.J. Popek, and J.S. Heidemann. Architecture of the Ficus scalable replicated file system. *Technical Report CSD-910005, UCLA,* USA, March 1991.

[17] M. Mauve and V. Hilt. An Application Developer's Perspective on Reliable Multicast for Distributed Interactive Media. In *Computer Communication Review*, pages 28-38, 30(3), July 2000.

[18] D.H. Ratner. Selective Replication: Fine grain control of replicated files. *Master's thesis, UCLA,* USA, 1995.

[19] G.H Kuenning. Seer: Predictive File Hoarding for Disconnected Mobile Operation. *PhD. dissertation, UCLA-CSD-970015. UCLA,* USA, 1997.

[20] C. Griwodz. Wide-Area True Video-on-Demand by a Decentralized Cache-based Distribution Infrastructure. *PhD. dissertation*, TU Darmstadt, April 2000.

[21] J. Chung-I and M.A. Sirbu. *Distributed Network Storage with Quality-of-Service Guarantees.* web site http://www.ini.cmu.edu/~sirbu/pubs/99251/chuang.htm