OSLS01

"accepted for the 27th EUROMICRO Conference, Workshop on Multimedia and Telecommunication; Warsaw, September 4-6, 2001"

Replication with QoS support for a Distributed Multimedia System

Giwon On[1], Jens Schmitt[1], Michael Liepert[1], and Ralf Steinmetz[1,2]

1: Darmstadt University of Technology
Merckstr. 25 • 64283 Darmstadt • Germany

2: FhG-IPSI
Dolivostr. 15 • 64293 Darmstadt • Germany

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

Abstract

Replicating data and services at multiple networked computers increases the service availability, fault-tolerance and quality of service (QoS) *of distributed multimedia systems. In this paper, we discuss some relevant design and implementation issues of a replication mechanism for a distributed multimedia system medianode[1] which is a software infrastructure to share multimedia-enhanced teaching materials among lecture groups. To identify new replication requirements, we first study the characteristics of presentational media types which are handled in medianode, then extract new replica units and granularities which have not been considered and not supported in existing replication mechanisms. Based on the new requirements and the result of feature surveys, we implemented a replication mechanism for medianode. The next working step is to evaluate the efficiency of our replica maintenance mechanism.*

1. Introduction

For practical use of a distributed multimedia system such as medianode[1] in a multimedia-enhanced teaching environment in which a fast and consistent accessibility of the teaching material for all accepted users of the system should be provided, the availability of the material must be increased by bypassing a variety of potential error sources. Replication of presentation materials and meta-data is a fundamental technique for providing high availability, fault tolerance and quality of service (QoS) in distributed multimedia systems[2], 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 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.

In this paper, we discuss some relevant design and implementation issues of a replication mechanism for a distributed multimedia system medianode[1] which is currently developed as an infrastructure to share multimedia-enhanced teaching materials among lecture groups. With the replication mechanism, medianode provides enhanced access to presentation materials in both connected and disconnected operation modes.

The structure of the paper is as follows. In Section 2, we identify new replication requirements. After analyzing the characteristics of presentational media types, we classify three different types of target replicas according to their granularity (data size), requirement of QoS support, update frequency. Section 3 presents the design and implementation issues for our replication model. We describe the proposed replication maintenance mechanism, e.g. how and when replicas are created and how the updates are signalled and transported. In Section 4, 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. 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. Identifying New Replication Requirements

2.1. Different Types of Presentation Data

In medianode, data organization comprises the storage of content data as well as meta information about this content data in a structured way. For the purpose of QoS-based generation of presentation files and replication, the system resource usages information such as the used memory number of the loaded medianode components is collected and managed.

Table 1: Data categories and their characteristics in medianode

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

The typical data types which can be identified in medianode are the following:

- Presentation contents: this type of data comprises text, image, audio/video files and can be stored in file systems which should handle automatic data distribution and access, and also support the multimedia characteristics of this content type.
- Presentation description data, e.g. XML files.
- Meta-data of user, system, domain, and organization information. User's title, group, system platform, and university are examples for this meta-data category.
- Meta-data of system resource usage information such as memory usage, number of threads running within medianode process, number of loaded bows.
- Meta-data of user session and token information.

- Meta-data of user, system, domain, and organization information. User's title, group, system platform, and university are examples for this meta-data category.
- Meta-data of system resource usage information such as memory usage, number of threads running within medianode process, number of loaded bows.
- Meta-data of user session and token information.

Table 1 shows an overview of these data types with their characteristics.

2.2. Classification of Target Replicas

The main goal of our replication system is to increase the 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, as shown in 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:

- Metareplicas (replicated metadata objects) that are persistent and of small size. An example would be a list of medianodes (sites) which currently contains 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 these 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.
- 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, to which the end users have access for direct manipulation (updating). On the other

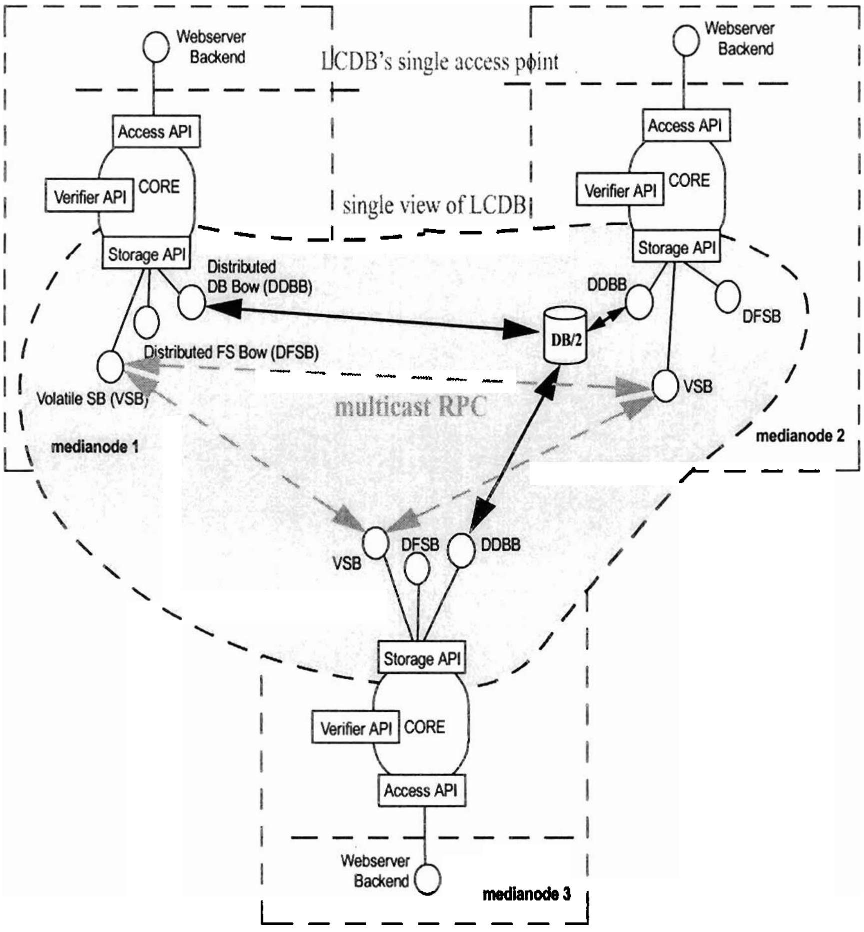

Figure 1: medianode architecture with replication service

side, these are also the only replica type which requires the support of really high availability and QoS provision.

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 do not support any replication service for function calls, and elementary data types.

2.3. Concept of Logically Centralized Database

For a technical realization of our proposed replication system in medianode, we use the concept of a so-called "logically centralized database (LCDB)" which especially enables the transparent access to presentation materials. Similar to the concept of location-independent identifiers in distributed database system[3], LCDB enables a mapping between logical and physical resources. So users do not need to know where presentation resources are located physically and how they are accessed. Requests from users, either for reading or writing any presentation materials, are first sent to the Access Bow of the local medianode that runs on the user's local machine. After successful check of the accessibility for the user and the availability of the requested resources, the corresponding storage bows send the target data to the users. Figure 1 illustrates the interface point, the bows building the LCDB and the interactions between the bows. Some additional remarks on LCDB are in order:

- According to the data types, all of the presentation contents and their meta-data are stored in corresponding storage bows.
- The 'front-end' of the storage bow API provides unique interface functions, independent of the data types: this is similar to the VFS (virtual file system) interface in UNIX systems.

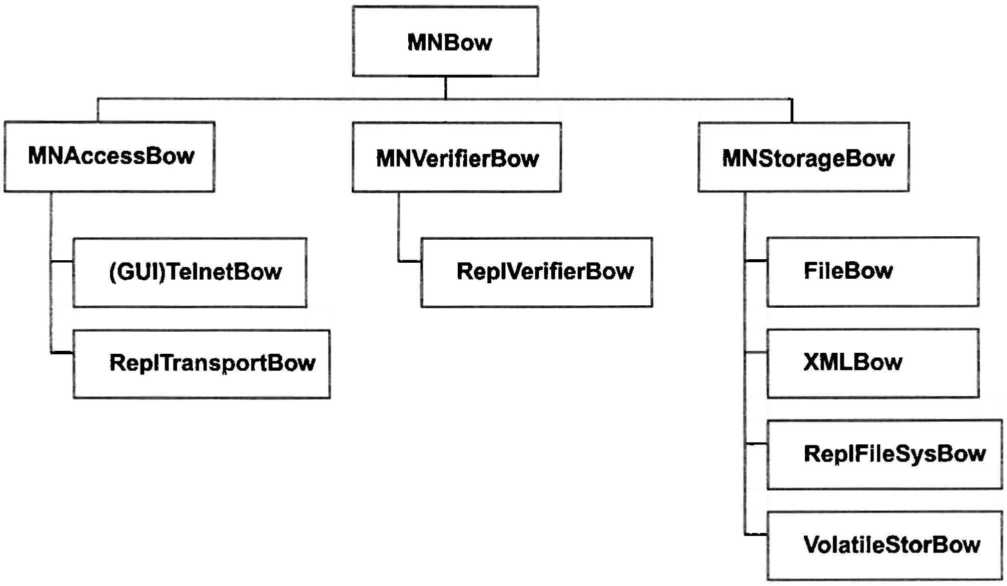

Figure 2: The medianode bow (MNBow) class hierarchy

- Replication has to be supported for most storage bows, although the number of replicas and the update frequency may differ between the individual bows.
- For the update propagation between replication managers, a multicast RPC (remote procedure call) communication mechanism is used.

3. Design and Implementation Issues

3.1. Scope of our Replication System

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. A replication service within a medianode, i.e., 'intra-medianode', is not considered for the first stage of our implementation. However, the replication concept in this paper is straightforwardly applicable to the replication service for intra-medianode scope.

3.2. The Replication Mechanism

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 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 subgroups. Examples for such sub-groups are file directories or a set of presentations about the same lecture topic.

3.3. Implementation Architecture

To show a 'prove of concept', we have implemented a prototype of the proposed replication system model for Linux platform (Suse 7.0, Redhat 6.2). Implemented are the 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 Volatile storage bow which maintains user's session and token information. Figure 2 shows a class hierarchy of medianode's basic bows and of extended bows for the replication system. MNBow is the root class and the three bow APIs, MNAccessBow, MNVerifierBow and MNStorageBow are implemented as MNBow's child class. [14] gives a detailed description of the implemented bows.

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. Based on this request-response mechanism, we experimented some presentation scenarios with and without a replication service.

3.4. Initialization of Replication Service

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 3, ReplFileSysBow sends a request packet via the core to ReplVerifierBow for creating a media list for media data which are located 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.

3.5. 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.

3.6. 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). 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 (step 6). The ReplVerifierBow then start a process to acquire a replica by creating a corresponding request packet which is passed to ReplTransportBow (steps 7~8). 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).

3.7. Update Distribution & Transport Mechanism

The update distribution mechanisms in medianode differs between the three replica types and their managers. This is due to the fact that the three replica types have different levels of requirements on and characteristics of high

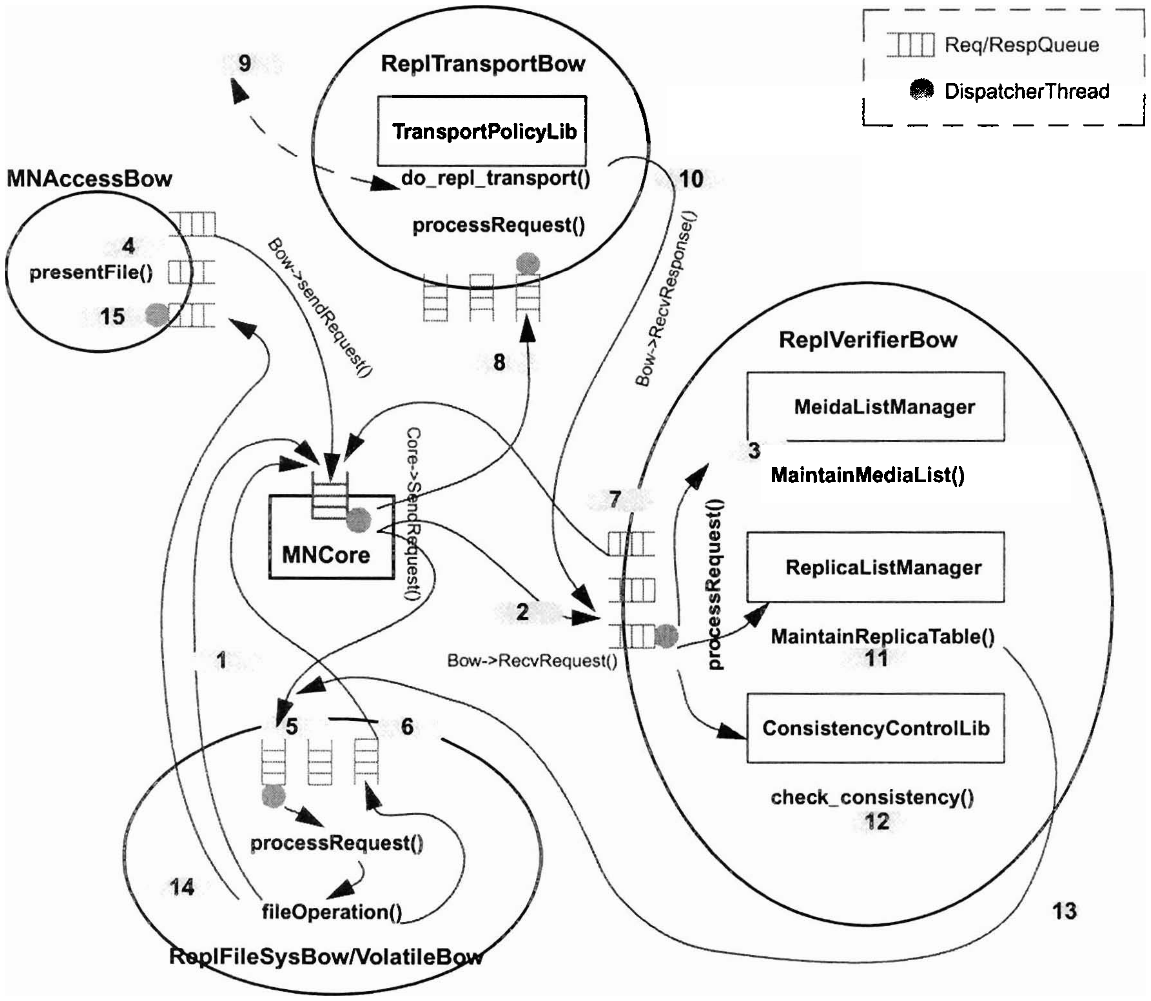

Figure 3: Service flow showing the internal bow interaction mechanism in medianode: with replication support

availability, update frequency and consistency. Experience from [4] and [5] also shows that differentiating update distribution strategies makes sense for web and other distributed documents.

The medianode's replication system offers a 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. The update transport and signalling protocols used are:

- RPC protocol [2] 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 [6,9] is used for the first implementation. RPC2 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.

3.8. Approaches for Resolving Update 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[8]. Depending on the used replication model and policy, there are different approaches to resolving update conflicts, of which our replication system will use the following strategies [2, 6, 7, 11]:

- 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 tentative update as a final update;
- Merging - to integrate two or more updates and build one new update table;

4. Related Work

There are many works and approaches to replication. 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 Coda[6], Roam[11], Rumor[13] and Ficus[17] 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 latency. 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 [6, 8, 15, 16]. 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[12]. 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[8] 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.

5. Summary and Future Work

Replication of presentation materials and meta-data is a fundamental technique for providing high availability, fault tolerance and quality of service (QoS) in distributed multimedia systems. In this paper, we discussed some relevant design and implementation issues of a replication mechanism for a distributed multimedia system medianode. After analyzing the characteristics of presentational media types, we classified three different types of target replicas according to their granularity, requirement of QoS support, update frequency. We also described the proposed replication maintenance mechanism, e.g. how and when replicas are created and how the updates are signalled and transported.

We are currently in the process of implementing the conflict resolving mechanism and versioning and storage/transport load levelling mechanisms, which are integrated with the replication manager. With the forthcoming implementation we will be able to build medianode as a highly available, scalable and cooperative, distributed media server for multimedia-enhanced teaching. The next working steps are to evaluate the efficiency of our replica maintenance mechanism and to design other replication service components. We are intensively investigating for the following issues for extension of our replication system:

- Reliable multicast-based update distribution mechanism: in the multicast-based replication environment, the replicas and their updates should be propagated 100% correctly to avoid any inconsistency between replicas. Although the RPC2 offers the multicast-based transmission, it does not guarantee any reliable transport of updates. LC-RTP (Loss Collection RTP)[10] is one of reliable multicast protocol which is originally developed as an extension of RTP protocol to support the reliable video streaming within the medianode project. We adopt LC-RTP and check the usability of the protocol, depending on the degree of reliability required for the individual groups of replicas.

- QoS-aware replication for distributed multimedia systems: the decision problems of (a) whether a replica should be created from original file and if then which files should be replicated (replica selection problem) and (b) to which system replicas should be put (replica placement problem) are made by checking the current usages of available system resources. [18] gives a survey on the works related two these problems and their performance models.

References

The medianode project. (http://www.httc.de/medianode).

G. Coulouris, J. Dollimore and T. Kindberg. *Distributed Systems*, 3rd Ed., Addison-Wesley, 2001.

A. Eickler, A. Kemper and D. Kossman. Finding Data in the Neighborhood. In *Proc. of the 23rd VLDB Conference*, Athens, Greece, 1997.

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

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.

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.

J. Yin, L. Alvisi, M. Dahlin and C. Lin. Volume Leases for Consistency in Large-Scale Systems. In *IEEE Transactions on Knowledge and Data Engineering*, 11(4), July1999.

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.

[9] 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.

[10] M. Zink, A. Jones, C. Girwodz and R. Steinmetz. LC-RTP (Loss Collection RTP): Reliability for Video Caching in the Internet. In *Proceedings of ICPADS'00: Workshop,* pages 281-286. IEEE, July 2000.

[11] 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. (web site http://lever.cs.ucla.edu/project-members/reiher/available_papers.html)

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

[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. (web site http://lever.cs.ucla.edu/project-members/reiher/available_papers.html).

[14] G. On and M. Liepert. Replication in medianode. *Technical Report TR-2000-03*, Darmstadt University of Technology, Germany, September 2000.

[15] C. Griwodz. Wide-Area True Video-on-Demand by a Decentralized Cache-based Distribution Infrastructure. *PhD. dissertation*, Darmstadt University of Technology, Germany, April 2000.

[16] 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.

[17] 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.

[18] M. Nicola and M. Jarke. Performance Modeling of Distributed and Replicated Databases, in *IEEE Transactions on Knowledge and Data Engineering*, 12(4), pages 645-672, July/Aug. 2000.