Data Replication in P2P Collaborative Systems

Fatos Xhafa
Technical University of Catalonia
Barcelona, Spain
fatos@lsi.upc.edu

Vladi Kolici
Polytechnic University of Tirana
Tirana, Albania
vkolici@fti.edu.al

Alina-Diana Potlog
University Politehnica Bucharest
Bucharest, Romania
dianapotlog@gmail.com

Evjola Spaho,
Fukuoka Institute of Technology
Fukuoka, Japan
evjolaspaho@hotmail.com

Leonard Barolli
Fukuoka Institute of Technology
Fukuoka, Japan
barolli@fit.ac.jp

Makoto Takizawa
Seikei University
Tokyo, Japan
makoto.takizawa@st.seikei.ac.jp

***Abstract* – Data replication techniques have been extensively used in distributed systems to achieve, among others, due to system nodes failures: (a) high data availability; (b) system's reliability and (c) scalability. Due to the various characteristics of distributed systems as well as system's and application's requirements, a variety of data replication techniques have been proposed in the distributed computing field. One important distributed computing paradigm is that of P2P systems, which distinguish for their large scale and unreliable nature. The study and application of replication techniques becomes a must in such systems. While it is well understood and easy to achieve replication of immutable information (typically files) in P2P systems, it becomes more challenging to implement data replication techniques of dynamic data under highly dynamic nature of large P2P systems. Indeed, replicating documents that could change over time requires addressing the consistency issues. In this paper we study some data replication techniques for P2P collaborative systems. We identify several contexts and use cases where data replication can greatly support collaboration. We then consider as a case study replication techniques for dynamic documents in the context of a peer-group based P2P system of super-peer architecture. P2P collaborative systems arise in many groupware applications, such as collaborative work in online teams, having requirements for high availability and system reliability (e.g. in disaster management scenarios). We propose a replication system for documents structured as XML files to address the dynamics of the documents at peers and use the super-peer to ensure a satisfactory level of document consistency among peers.**

Keywords – Data Replication; P2P Collaborative Systems; JXTA Library; Peer-group; Super-peer, XQuery

I. INTRODUCTION

Data replication techniques have been extensively used in distributed systems as an important effective mechanism for storage and access to distributed data. Data replication is the process of creating copies of data resources (data, multimedia content, ...) in a network. Data is replicated at more than one site, creating replicas of the original objects. Obviously, data replication is not just copying data at multiple locations as it has to solve several issues involved!

Data replication can improve the performance of a distributed system in several ways, basically due replication creates data redundancy:

1) *High availability, reliability and fault tolerance*: data availability is increased and thus avoid service disruption due system nodes failure.
2) *Scalability*: the service capacity is increased due server loads can be decreased. Response time and *QoS* requirements can be greatly improved.
3) *Performance*: Increased performance due data access (especially when the requests involve only reads).
4) *"Fail Safe"* infrastructures: replication is a choice for today's critical IT systems as replication is key to reducing the time for service recovery.

Due to the various characteristics of distributed systems as well as system's and application's requirements, a variety of data replication techniques have been proposed so far in the distributed computing field.

One important distributed computing paradigm is that of P2P systems, which distinguish for their large scale and unreliable nature. The study and application of replication techniques becomes a must in such systems to achieve properties 1)—4) above. While it is well understood and easy to achieve replication of immutable information (typically files) in P2P systems, it becomes more challenging to implement data replication techniques of dynamic data under highly dynamic nature of large P2P systems. Indeed, replicating documents that could change over time requires addressing the consistency issues. Consider for instance a group of peers that work together to accomplish a common project. One main requirement is that peers must have access to the project documents all the time, while these documents are distributed at different peers. Certainly, this requirement cannot be satisfied due peers can go on and off during the time. It is through the replication that such availability issue can be addressed. However, even if the data and documents' availability were ensured, project documents can involve more than a peer

978-0-7695-4841-8/12 $26.00 © 2012 IEEE
DOI 10.1109/3PGCIC.2012.26

working on that and documents can change over time. The document consistency becomes than a main issue.

In this paper we study some data replication techniques for P2P collaborative systems, viewed as a family of techniques and classified under different criteria (full *vs.* partial, synchronous *vs.* asynchronous, optimistic, lazy *vs.* pessimistic, push *vs.* pull update modes, etc.). We identify several contexts and use cases where data replication can greatly support collaboration. We then consider as a case study replication techniques for dynamic files in the context of a peer-group based P2P systems of super-peer architecture. P2P collaborative systems arise in many groupware applications, such as collaborative work in online teams, having requirements for high availability and system reliability (e.g. in disaster management scenarios). We propose a replication system for documents structured as XML files to address the dynamics of the documents at peers and use the super-peer to ensure a satisfactory level of document consistency among peers.

The rest of the paper is organized as follows. We present some contexts and uses where data replication techniques arise in Section II. The main data replication techniques in P2P systems are briefly given in Section III. In Section IV we focus on the case of P2P collaborative systems, for which a replication system for documents structured as XML files, with super-peer architecture is proposed. We address the consistency of the replication system when using optimistic replication techniques. The system implementation is presented in Section V. We conclude the paper with some remarks and directions for future work in Section VI.

II. DATA REPLICATION: CONTEXTS AND USES

Data replication arises in many contexts of distributed systems and applications.

A. Distributed storage

One main context of replication is that of Distributed Database Management Systems (DBMS). With the emergence of large scale distributed computing paradigms such as Cloud, Grid, P2P, Mobile Computing, etc. the data replication has become a commonplace approach, especially, to ensure scalability to millions of users of such systems. In particular, data replication is used in Data Centers as part of Cloud Computing systems.

B. Disaster management scenarios

In these scenarios, ensuring anytime access to data is a must. The rescue teams need to collaborate and coordinate their actions and anytime access to data and services is fundamental to support teamwork because decision taking is time-sensitive and often urgent.

C. Business applications

Data replication has attracted the attention of researchers and developers from businesses and business intelligence as a key technique to ensure business continuity, continuity-of-operations, real time access to critical data as well as for purposes of handling big data for business analytics (see Sheppard [11] and van der Lans [12]). In such context replication is seen as a choice to make data in a business environment operational.

D. Collaborative and groupware systems

One important requirement in collaborative and groupware systems is to support distributed teamwork, which often suffers from disruption. For example, supporting large user communities (e.g. from High Energy Physics community) during scientific collaborations projects. Replication is thus a means to ensure access to data anytime and support thus collaboration even in unreliable networking environments. This later feature is each time more important due to the mobility of the teamwork and use of mobile devices.

III. DATA REPLICATION TECHNIQUES IN P2P SYSTEMS

A. Setting up a replication plan

Implementing data replication requires setting up a replication plan and to answer some key questions to ensure desired properties of the system.

1. *What to replicate?* This is to identify the kind of data to replicate.
   - Full objects, fine-grained objects, chunks/blocks can be replicated.
   - Data could be documents, files, meta-data, multimedia, user profiles, events, messages, etc.
   - Data could be static or dynamic over time.
   - Replication could be meant for access purposes only or for disaster recovery as well.
   - Evaluate the homogeneity/heterogeneity degree of the data (heterogeneous *vs*. homogenous data).
   - Evaluate the degree of structuring of the data (structured *vs*. unstructured data)
2. *Where to replicate?* This question has to do with the underlying computing environment where replication will take place.
   - Evaluate the heterogeneity degree of the computing environment (heterogeneous *vs*. homogenous computing environment).
   - Evaluate how much storage capacity, performance and reliability, type of storage available at replicated sites.
   - Evaluate the cost of the replication in the underlying infrastructure.
3. *How to replicate?* This should address the needs of:
   - How much data has to be online (to be replicated)? Will the replication be done synchronously or asynchronously?
   - How should be related the original data and its replicas? Decide the consistency type, full vs. partial replication, etc.

B. Consistency and limits to replication

Replication should be transparent to the user, it has to achieve one logical view of the data! The fact that all users see the same data at any time is expressed in terms of consistency. Full consistency means that original data and its replicas are identical, while in partial consistency state there are differences or conflicts among original data and its replicas. One main issue is thus to achieving a satisfactory degree of consistency so that all users see the same data. The degree of consistency depends on many factors, but primarily it depends on whether the application or system can tolerate a partial consistency. Based on this, there have been defined:

- *Pessimistic data replication*: this kind of replication achieves fully consistent data and avoids conflicts, but reduces the responsiveness of the system. If applied to a collaborative system, it would limit the collaboration.
- *Optimistic data replication*: this kind of replication allows conflicts to happen (although they are not frequent), tries to fix them when they happen based on operation relationships. The resulting consistency state is called eventual consistency. This replication is usually implemented through lazy strategies, which impose minimum requirements on the communication environment (as opposed to pessimistic/eager strategies). In several collaborative systems, especially those based in a asynchronous collaboration, this kind of consistency is tolerable due the pattern of communication among team members enables the system to become consistent.

Consistency: Brewer's conjecture –CAP Theorem: Brewer's conjecture (Eric Brewer, *"Towards Robust Distributed Systems"*, 2000), later proved by Gilbert/Lynch in 2002 for asynchronous model [10] states that *"Distributed web services cannot fulfill at the same time three properties: Consistency, Availability and Partition tolerance"*:

- *Consistency*: all nodes read the same data at any point in time.
- *Availability*: a node failure does not cause failure in other nodes/systems.
- *Partition tolerance*: The system continues to function despite any number of message/node failures.

This result, known as CAP theorem, puts a theoretical limit as to what can be achieved by distributed systems. Based on this result a more practical formulation has been proposed by Pritchett in 2008 *"Basically Available, Soft estate, Eventually consistent"*, which states the need for trading consistency for availability.

Data replication techniques can be classified using three criteria: where updates take place (single-master *vs*. multi-master), when updates are propagated to all replicas (synchronous *vs*. asynchronous) and replicas' distribution over the network (full *vs*. partial replication). We briefly describe them next (see also Xhafa *et al.* 2012 [13], [14]).

C. Single-master vs. multi-master

The single-master model allows only one site to have full control over the replica (read and write rights) while the other sites can only have a read right over the replica [6]. One of the advantages of this model is that it avoids concurrent updates at different sites during the process of centralizing updates. Moreover, it assures that only one site holds the latest version of the replica. The drawbacks of this model are that the master becomes a single point of failure and that there is the chance of master bottleneck. In case of master failure, no updates could be propagated in the future and this reduces data availability. The bottleneck appears when the master has to manage large number of replicated documents.

The single-master approach is depicted in Fig. 1. The updates can be propagated through *push mode* or *pull mode*. In *push mode*, it is the master that initiates the propagation of the updates, while in the case of *pull mode*, the slave queries the master for existing updates.

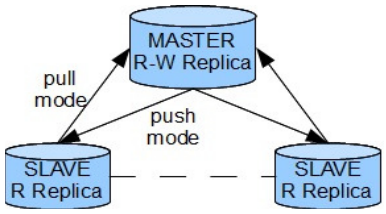

Figure 1. Single-master replication

The multi-master model allows multiple sites the right to modify their saved replica (Fig. 2). The system is responsible with propagating updates operated by a member of the group to all the other members and to solve any conflicts that may appear between concurrent changes made by different members [6]. Replica reconciliation can be a cumbersome and complicated process and, in most cases, the replicas are loosely consistent. On the other hand, propagating updates from different sites in the group can lead to expensive communication. The multi-master model is more flexible than the single-master model as, in case of one master failure, other masters can manage the replicas.

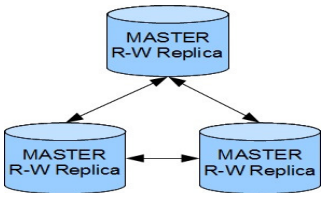

Figure 2. Multi-master replication

D. Synchronous vs. asynchronous replication

These replication models use the notions of transaction, commit and abort. A transaction is a set of update operations (writes). If a transaction is committed, that means that all the update operations that compose the transaction are

performed on an object, if there are no errors. In case of errors a transaction is aborted. In a replicated system a transaction that updates one replica must be propagated to the other replicas in order to provide replica consistency. The update propagation can take place in two ways: the node that initiated the transaction waits for the data to have been recorded at the other nodes before finally committing the transaction (synchronous replication) or the node commits the transaction locally and afterwards propagates the changes (asynchronous replication).

In *synchronous replication*, the node that initiates the transaction propagates the update operations within the context of the transaction to all the other replicas before committing the transaction (Fig. 3). Thus, this mechanism guarantees atomic write (data is written at all sites or at none). There are several algorithms and protocols used to achieve this behavior: two-phase-locking (2PL), timestamp based algorithms, two-phase-commit protocol (2PC) [1].

The above protocols bring some performance limitations – transactions that need to acquire resource locks must wait for resources to be freed if these are already taken. Kemme and Alonso [2] proposed a new protocol to avoid the drawbacks of the above protocols. It takes advantage of the rich semantics of group communication primitives and the relaxed isolation guarantees provided by most databases.

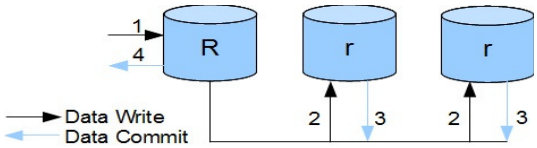

Figure 3. Synchronous replication

Asynchronous replication does not change all replicas within the context of the transaction that initiates the updates. The transaction is first committed at the local site and then updates are propagated to remote sites (Fig. 4). The asynchronous replication can be classified as optimistic or pessimistic in terms of conflicting updates. The optimistic approach assumes that conflicting updates will take place rarely [6]. Updates are propagated in the background and conflicts are fixed after they happen. The pessimistic one assumes that conflict updates are likely to occur and implements mechanisms for replica consistency [3][4].

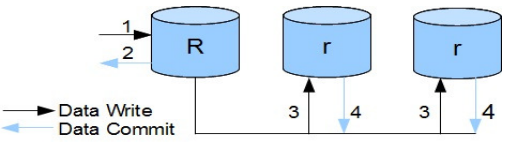

Figure 4. Asynchronous replication

With asynchronous replication, performance is greatly increased as there is no locking anymore. But if the node that initially updates the replica is down then all the other nodes will not have the up-to-date values of the replica.

E. *Full vs. partial replication*

Placing a replica over the network has an impact on replica control mechanisms. There are two main approaches for replica placement: full replication and partial replication.

Full replication takes place when each participating site stores a copy of every shared object. This requires each site to have the same memory capacities and maximal availability as any site can replace any other site in case of failure. In the case of *partial replication*, each site holds a copy of a subset of shared objects so that sites can keep different replica objects. This type of replication requires less storage and reduces the number of messages to update replicas since updates are propagated only to the interested sites. A reduced load is produced for the network and sites but the propagation protocol becomes more complex since the replica placement must be taken into account. Moreover, this approach limits load balance possibilities due certain sites are not able to execute a particular type of transaction [5]. In partial replication, it is important to find a right replication factor. Careful planning should be done when deciding which documents to replicate and at which peers.

F. *Replication's operations and optimistic replication*

Operations. An operation is an update to an object. Operations are similar to transactions performed in the traditional databases except that they are also propagated and applied in the background, many times after they were submitted by the users. There are two ways to propagate updates taking into account the storage of operations. One way is to store operations in log files and then propagate them to remote sites to assure replica consistency. These kind of systems are called operation-transfer systems. Examples of such systems are Bayou and IceCube. Another way is to propagate the current state of the object. Such systems are called state-transfer systems. DNS, Unison and Harmony are included in this category.

Operation Relationships. They represent implicit or explicit associations between operations. Conflicting updates are detected based on relationship between operations and are then arranged in a convenient order. There are four types of operation relationships that are relevant for optimistic replication systems: happens-before, concurrency, explicit constraint and implicit constraint.

Propagation Frequency. Propagation happens when operations and replica states are send to remote sites in order to achieve the consistency of the replica. The frequency of propagation is determined by the strategies pulling, pushing or hybrid. Pull-based systems update their local copies by pulling other remote sites either on demand (e.g. CVS) or periodically (e.g. DNS). In push-based systems sites send their updates to the other sites as soon as possible (e.g. LOCUS). Hybrid systems combine the first two strategies (e.g. TSAE). In pull-based, replicas arrive faster to a consistent state.

Conflict Detection and Resolution. When there is no site-coordination multiple users can update the same replica at the same time thus leading to possible conflicts. Conflicts must be detected and then eliminated using resolution approaches. Conflict detection policies are classified as none,

concurrency-based and semantic-based. With none policy conflicts are ignored. But this policy leads to lost updates. Systems with concurrency-time policy declare a conflict between two operations based on timing of operation submission (e.g. LOCUS). Systems that are aware of operations' semantics can reduce conflicts (e.g. Bayou, IceCube). Conflict resolution is highly application specific and can be either manual or automatic. Manual approach is adopted by the majority of the systems and requires user intervention to fix it (e.g. CVS). But there are systems that solve the conflict automatically (e.g Bayou).

Reconciliation. A certain replica can be modified at different sites, thus permitting applications to continue functioning even if some sites are are offline or down. This kind of parallel updates can cause replica divergence. The process of reconciliation brings these replica to a mutual consistent state. There are different reconciliation strategies that depend on the type of input information and the criterion for ordering updates. The input information can be the current states of the replicas or the update operations. State-based reconciler is a reconciliation engine that takes as input the updated states of the replicas and tries to make them as similar as possible (e.g. Harmony, Unison). Operation-based reconciler is a reconciliation engine that accepts as input the history of operations performed at each replica and tries to build a common sequence of operations (e.g. Bayou and IceCube). The criterion for ordering updates can be based on ordinal information associated with updates or on semantic properties. The ordinal-reconciler tries to maintain the submission of updates based on when, where and by whom updates were performed.

This is achieved using timestamp-based ordering (e.g. TSAE) or version vectors (e.g. LOCUS). The semantic-reconciler exploits semantic properties associated with updates to reduce conflicts (e.g. IceCube).

IV. Replication in P2P Collaborative Systems

P2P collaborative systems are P2P based groupware systems. In such systems, peers are grouped into peergroups. Within a group, peers are equal nodes, yet they permit client/server configurations when needed. Peers have their replica repository and implement rule engines (ECA= Event-Condition-Action rules). Similarly, super-peers implement ECA rule engines. Unlike other large scale distributed systems such as Server-based systems, P2P collaborative systems distinguish as highly heterogeneous computing environments in terms of computational and network resources and highly dynamic and unreliable computing environment. Additionally, peers could show different behaviors within the peergroup such as cooperative, volunteer contribution, coalition, selfish and free riding behavior. Nevertheless, P2P model offers a suitable and robust operational environment for large families of collaborative applications. Clearly, existing solutions from DBMS for highly reliable networks do not fit well for real-time data replication and for P2P networks.

We consider P2P collaborative systems with a super-peer architecture.

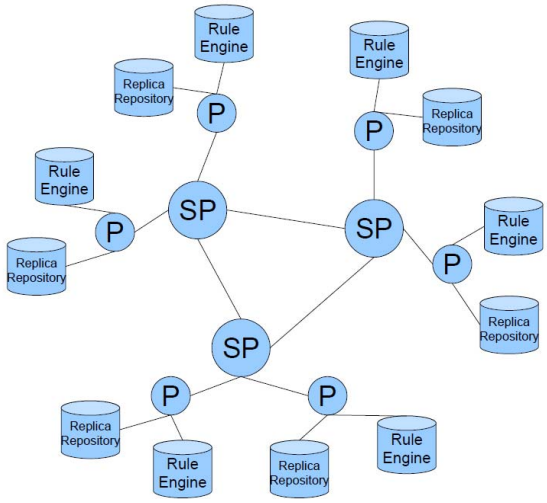

Figure 5. Super-peer architecture with disjoint peergroups

In our context, a group of peers work together to accomplish a joint project. A group-project consists of several tasks that are to be completed by the peers in the group. Each task implies document replication, document updates and convergence of replicas among several peers from the group. The description of the group-project is a table similar to the one below:

Table 1: Group project description.

Task			Assigned Peers	Start date	End date	Task Precedence (dependencies)	Documents (associated to task)
Id	Name	Description					
7	Revision	Revision of the design document	P1, P2	date1	date2	Requires completion of tasks Id 4, 6	D1, D3

Different from existing approaches, where mostly static documents are replicated, we implement a fine-grained document description using XML document representation. Indeed, in teamwork for a common project usually changes to some parts of a document are done; therefore, we only need to update some parts of the corresponding replica. Specifically, changes in a node of the XML tree are propagated to the peer(s) holding the respective replica(s), which in turn will update the replica(s).

The proposed replication system uses optimistic replication techniques with propagating update operations from source node to destination node in push mode. It is suitable for asynchronous collaboration and works for both full and partial replication.

A. *Peer-group design*

The peer-group is organized in such a manner that each peer can directly communicate with any other peer in the group (see Fig. 6). Furthermore, the peer-group has a central manager, the super-peer, which has two important roles. One role regards only the peer-group in which the super-peer resides. The super-peer keeps track of the project development and assigns tasks to the peers in the group. The other role regards the whole network. The super-peer

facilitates the communication with other peers in other peer-groups.

The simple peer-group structure is enhanced with a Data Repository (DR) that acts as a storage facility in order to keep the initial documents related to the project (Fig. 6). When a peer receives a task, it then connects to the DR to get copies of the documents which it can modify afterwards locally.

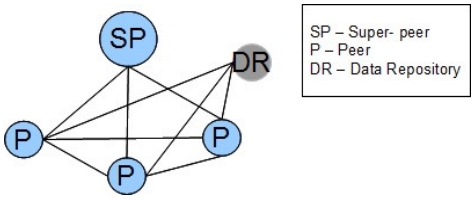

Figure 6. Peer-group structure with a Data Repository

We use JXTA library [8], [9] for prototyping our approach. In JXTA, a peer-group is uniquely identified by a *PeerGroupID*. We design our peer-group with a custom membership policy that requires each peer credentials (user and password) when joining the group. The communication within the members of the group is done using unicast and multicast communication.

B. *Peer-group entities*

Super-peer. A super-peer acts as a server for some peers, namely the peers found in the same group with it, and as an equal in a network of super-peers (Fig. 5). The advantage of having a super-peer is that it can greatly improve the consistency control. The super-peer in peer-group maintains global advertisement indices and assists edge peers with advertisement searches. This means that the super-peer is aware about the other super-peers in the network and thus it makes the connection between the peers in the group and the other peers and super-peers from the network.

The super-peer is responsible for assigning tasks, storing all updates in the system and answering query requests from clients requesting information about the latest versions in the system. The super-peer must send requested versions to clients and is also responsible with erasing old updates.

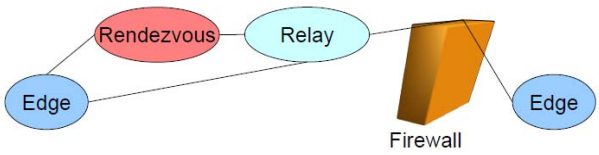

Figure 7. Peer types in JXTA network

Peers. Peers are responsible for solving project's tasks. After the peer receives a task from the super-peer, it then contacts the DR to download the files related to the received task. Then, the peer will operate changes only on the documents that it has the right to modify. Peers need to propagate the changes made to the documents within the group in order to ensure replica consistency. In order to provide that late joining peers arrive at a consistent state rapidly, the peers must first send updates to the super-peer and thus late joining peers can query the super-peer and retrieve all the updates missing.

We design two versions of peer propagation behavior. One version uses multicast in order to propagate the changes to members of the group while the other version uses direct communication.

Data Repository. Data Repository is a logical and possibly physical partitioning of data where original documents reside and where final versions of the documents are saved. The repository can be kept at super-peer site or at another location in the peer group, possibly another specialized node.

C. *Peer propagation behavior*

Multicast Propagation. The multicast version is suitable for small write logs with a maximum size of 64KB (the maximum size of a multicast datagram). As the multicast is unreliable, it is better to transmit an update using one datagram in order to avoid extra work at peer site involving ordering of datagrams that contain chunks from the same write log. Peers don't need to store presence information about other peers. Each peer is responsible for filtering the updates received. Multicast also ensures fast delivery when transmitting small logs. But multicasting might lead to lost packets and network flooding. In the case of partial replication, all peers in the group receive the same updates. Multicasting is not scalable for large write logs.

Unicast Propagation. The unicast solution is suitable for large write logs. It guarantees reliable transmission, no packet is lost. It works for both partial and full replication, without flooding the network unnecessary in the case of partial replication. The drawback with unicast propagation is that peers receive updates with a certain delay and some peers will already have received the update while others are still expecting it to arrive. Unicast propagation does not scale in its primitive form, where no network topology is established.

D. *Communication between entities*

Peers communicate with the super-peer through sockets. This is because sockets provide reliable transmission of data and there is no need for supplementary verification at peer site. The peer sends task requests, task termination notifications, push update operations to and pull updates from the super-peer. The peer is also responsible with transmitting the current version of documents in response to query requests asking for it sent by the super-peer using *JXTA Peer Resolver Protocol* [8], [9].

The interactions between peers and super-peer are depicted in Fig. 8.

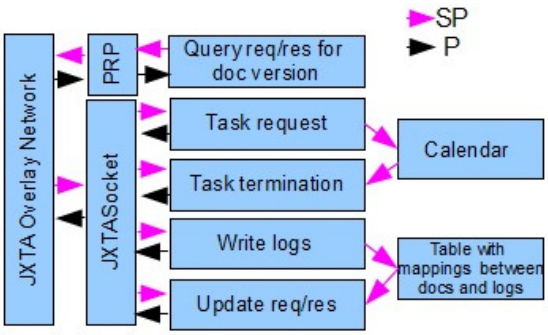

Figure 8. Peer-Super peer interaction

The *Calendar* structure keeps track of the progress of the project: tasks that have been submitted to peers, which peers finished a certain task, what tasks are finished by all peers, what tasks are in stand by, etc.

The super-peer receives every update operation from online peers. An update operation consists of several write operations. We call the set of write operations a write log. The super-peer keeps in a table the mappings between document IDs and the corresponding write logs. This table is structured in the form of a hash table, where the keys are IDs of the documents that are replicated among the peers of the group and and the values are vectors of write logs, sorted by the version of the update. The super-peer can thus respond to update requests issued by late joining peers.

In order to avoid storing thousands of updates at the super-peer and thus risking the super-peer to run out of memory, it is necessary that the super-peer queries periodically the peers in the group asking for their last versions of the documents. Thus, the super-peer uses the Peer Resolver Protocol [8], [9] in order to send queries requesting the last version of the documents stored at the peer nodes. The super-peer will then delete part of its logs only if all the peers in the group transmitted their last version of the documents.

Most part of the communication between super-peer and peers is realized through *JXTASocket* [8], [9] enabling that super-peer can handle multiple connections simultaneously. The super-peer accepts the following types of connections from the peers: for requesting a task, for task termination, for transmitting changes and for requesting updates. The threads created for *Task request* and *Task termination* operate modifications in the task entry from the calendar object. The thread created for *Write logs* received from clients adds the received log with to the *Table with mappings between docs and logs*. The thread created for *Update request* retrieves logs with versions greater than the ones stored at the requesting peer and sends those logs to the peer. When a task finishes, all replicas are in a final state.

The interaction of super-peer with peers is depicted in Fig. 9.

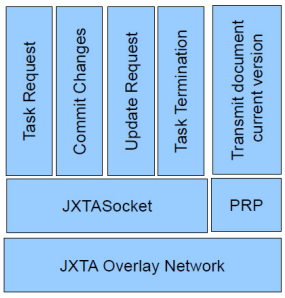

Figure 9. Communication of super-peer with peers

P2P Interaction. The interaction between peers uses multicast communication to propagate changes (update operations) (Fig. 10(a)). The current version of the document is propagated together with the changes. The interaction between peers using unicast is depicted in Fig. 10(b). The peer who has updates to propagate, must first use the *JXTA Discovery Service* [8], [9] to find the list of peers that are currently in the group, then pushes the changes using unicast communication to the discovered peers.

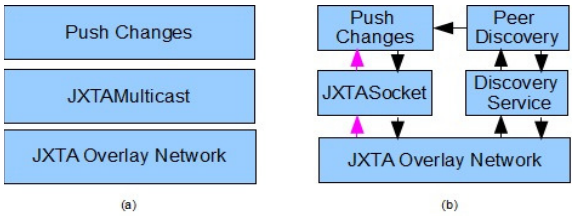

Figure 10. Interaction between peers: (a) multicast, (b) unicast

Peer – Data Repository Interaction. The peer uses the *Content Management Service* offered by JXTA to download files from a specified location [9] (see Fig. 11).

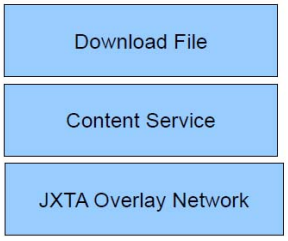

Figure 11. Peer Interaction with the Data Repository

V. SYSTEM IMPLEMENTATION

We detail in this section the protocols implemented for the communication between peers and between peers and super-peer as well as the mechanism through which peers acquire the documents that they will change later during project's task solving.

A. Communication protocols

We present the protocols implemented for the communication between two peers and between a peer and the super-peer. These protocols are used in implementing the architecture described in Section IV.

Peer – Super-peer protocols: The communication between a peer and a super-peer uses the following protocols: *Task Request*, *Commit Changes, Update Request* and *Task Termination*. These protocols are represented in the form of message sent through sockets. They are initiated by peers. In the case of the protocols *Commit Changes* and *Task Termination* the super-peer is not supposed to reply. Each message sent by the peer to the super-peer is differentiated by the first byte of the message, which represents the code of the message.

Task Request: This protocol is used whenever a peer connects to the super-peer for requesting a task (the peer is said to be in pull mode). The peer usually requests tasks when it enters the network for the first time or after it finishes the current task and asks for new tasks to be assigned to it. The super-peer must answer in reply to the request received. It will send a new task if there are still tasks to be solved or signals the fact that there are no more unsolved tasks left.

Commit Changes: This protocol is used to send to the super-peer the changes of a document.

Update Request: This protocol is used in the moment a peer changes the status state from OFFLINE to ONLINE and through which the peer sends an update request to the super-peer. The update request contains the current versions of the documents that the peer already has.

Task Termination: This protocol is used whenever a peer finishes the task that has been assigned. The super-peer must know when a task finishes in order to be able to assign new tasks for the accomplishment of the project.

Peer – Peer Protocols: A peer communicates with other peers only when the first modifies a document and needs to commit changes to the collaborative peers. The protocol is *Commit Changes* and implies multicast communication. The message format is:

<docID version updates>
docID – ID of the document modified
version – the current version of the document
updates – log of writes. Each write is a pair node-value, where node is the document node that was modified and value is the new value of the node.

B. Peer and super-peer interfaces

The peer interface contains methods for the JXTA environment provides thread objects for status service, modifying documents, update notifications received from other peers and propagating local updates within the peer-group.

```
//Peer Interface:
// threads
PeerStatusService peerStatusService; //change the status of the peer
PropagateUpdates propagate; //thread for propagating updates
//within the group
EditDocs edit; //thread for editing documents
ReceiveUpdates recvUpdates; //multicast socket for receiving updates
//(write logs)
UpdateRequest updateReq; //request updates from the super-peer each
 //time the peer changes its status from OFFLINE to ONLINE
 //JXTA related
 launchJXTA(); //start JXTA
 getServices(); //retrieve JXTA services
 searchForGroup(); //connect to the peer-group
 //starting threads
 peerStatusService.start();
 propagate.start();
 edit.start();
 recvCommit.start();
 updateReq.start();
```

The super-peer interface contains methods for the JXTA environment and a thread object for receiving updates (write logs) and update request.

```
//Super-peer interface:
//thread
ReceiveReqests recvRequests;
//JXTA related
launchJXTA();
getServices();
searchForGroup();
//methods
recvRequests.start();
```

C. Super-peer cache

Super-peer stores all the write logs that the peers from the group are sending through multicast. Along peers' activity, the size of the cached logs can become very big and thus, in order to make space for the new logs, the old ones must be deleted. There are two alternatives. One is to assign a time to live (TTL) field to each write log and erase those logs whose TTL field reaches zero. In this case, there are peers that will loose some updates after rejoining the network. The second alternative is for the super-peer to periodically send queries requesting for the current versions of the documents stored by the peers in the group. If all the peers answer to the query, then the super-peer deletes from its cache all the updates already committed by all the peers.

D. Document Representation

We choose to use XML documents as document replicas. They are easy to manipulate using XML database systems. The documents we have used in the replication system were built using the following document type definition (DTD) schema associated with the documents:

```
<!ELEMENT document (part+)>
<!ATTLIST document id CDATA #REQUIRED>
<!ELEMENT part (#PCDATA)>
```

We have assumed this sufficiently fine fine-grained description so that any change can be automatically done to the replicas. This XML structure is transparent to replication techniques and fully depends on user design.

In order to easy modify the content of the XML documents we found it necessary to use a XML Database. We have chosen the BaseX XML Database because it is free, can be integrated with Java and has XQuery API [14]. Update operations on XML files mean changing the value of a certain node. For that, we have used the following Xquery syntax:

```
replace value of node /document/part[1] with new_value
```

We have used the following class for updating documents:

```
class DocUpdater:
DocUpdater(String docID)
//updates the node node with the value new_value
updateNode(String node, String new_value)
writeToFile()//writes the content of the XML database to disk
//executes an Xquery on the current document
query(final String query)
```

VI. CONCLUSIONS AND FUTURE WORK

In this paper we have presented data replication techniques for P2P collaborative systems. As in other types of large scale distributed systems, data replication is useful to achieve important system's properties due to system node failures: (a) high data availability; (b) system's reliability and (c) scalability. While it is well understood and easy to achieve replication of immutable information (typically files) in P2P systems, it becomes more challenging to implement data replication techniques of dynamic data under highly dynamic nature of large P2P systems. Indeed, replicating documents that could change over time requires addressing the consistency issues. In this paper we have studied some data replication techniques for P2P collaborative systems. We have identified several contexts and use cases where data replication can greatly support collaboration. We then considered as a case study replication techniques for dynamic files in the context of a peer-group based P2P systems of super-peer architecture. In our context, we considered a group of peers aiming to accomplish together a common project. It is the project's documents that are replicated at peers' sites to ensure peergroup's work continuity both offline and online. We have proposed an optimistic replication system for documents structured as XML files to address the dynamics of the documents at peers and uses the super-peer to ensure a satisfactory level of document consistency among peers. We have presented the communication protocols, using JXTA library, among super-peer and peers as well as among peers themselves to achieve the correct system implementation. Our system is aimed to support online teams in an asynchronous collaborative environment where conflicts are expected to happen in long intervals of time.

In our future work we would like to evaluate the system in a real environment and measure the degree of consistency achieved in a real team project setting. Based on our online students' profile, we plan to evaluate the suitability of optimistic replication techniques in a real Virtual Campus.

REFERENCES

[1] Philip A. Bernstein, Vassos Hadzilacos, Nathan Goodman, "*Concurrency Control and Recovery in Database Systems*", 1987

[2] Bettina Kemme, Gustavo Alonso, "A new approach to developing and implementing eager database replication protocols", *ACM Transactions on database systems*, 25(3): 333-379, 2000

[3] Esther Pacitti, Pascale Minet, and Eric Simon. 1999. "Fast Algorithms for Maintaining Replica Consistency in Lazy Master Replicated Databases". In *Proceedings of the 25th International Conference on Very Large Data Bases* (VLDB '99), Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie (Eds.). Morgan Kaufmann Publishers Inc., USA, 126-137

[4] Cedric Coulon, Esther Pacitti , Patrick Valduriez, "Consistency Management for Partial Replication in a High Performance Database Cluster", *Proceedings of the 11th International Conference on Parallel and Distributed Systems* (ICPADS'05), pp. 809-815, July 20-22, 2005

[5] Yasushi Saito and Marc Shapiro. 2005. "Optimistic replication". *ACM Comput. Surv*. 37, 1 (March 2005), 42-81.

[6] Vidal Martins, Esther Pacitti, Patrick Valduriez, "Survey of data replication in P2P systems", TechRep 2006

[7] Beverly Yang, Hector Garcia-Molina, "Designing a Super-Peer Network", *Proceedings of the 19th International Conference on Data Engineering*, 2003

[8] JXTA Java Standard Edition v2.5: Programmers Guide, 2007

[9] JXTA Java Standard Edition v2.6: Programmers Guide, 2010

[10] Nancy Lynch and Seth Gilbert, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services", *ACM SIGACT News*, Volume 33 Issue 2 (2002), pg. 51-59.

[11] Eric Sheppard. Continuous Replication for Business-Critical Applications. *White paper*, January 2012

[12] R.F. van der Lans. Data Replication for Enabling Operational Business Intelligence. *Whitepaper on Business Value and Architecture*, June 2012

[13] Fatos Xhafa, Alina-Diana Potlog, Evjola Spaho, Florin Pop, Valentin Cristea and Leonard Barolli. Evaluation of intra-group optimistic data replication in P2P groupware systems. *Concurrency and Computation: Practice and Experience*, Vol. 24 Issue 12, DOI: 10.1002/cpe.2836, 2012, Wiley

[14] Angel Navarro-Estepa, Fatos Xhafa, Santi Caballe, A P2P Replication-Aware Approach for Content Distribution in E-Learning Systems," In Proceedings of CISIS-2012, *Sixth International Conference on Complex, Intelligent, and Software Intensive Systems*, pp. 917-922, 2012

[15] XQUERY: http://basex.org/