Evaluation of Intra-Group Optimistic Data Replication in P2P Groupware Systems

Fatos Xhafa*,[1] and Alina-Diana Potlog[2] and Evjola Spaho[3] and Florin Pop[2] and Valentin Cristea[2] and Leonard Barolli[3]

[1] *Technical University of Catalonia, Spain. E-mail: fatos@lsi.upc.edu*
[2] *University Politehnica of Bucharest, Romania. E-mail: dianapotlog@gmail.com, florin.pop@cs.pub.ro, valentin.cristea@cs.pub.ro*
[3] *Fukuoka Institute of Technology, Japan. E-mail: evjolaspaho@hotmail.com / barolli@fit.ac.jp*

SUMMARY

P2P computing systems have become very popular during the last years due to their ability to scale to a large number of users and efficient communication among peers. They can support complex computational processes, beyond simple file sharing, while offering advantages of decentralized distributed systems. However, such systems may suffer from availability and reliability. In order to increase availability and reliability, and therefore, improve the perception of peers, yielding to fast response times and rich experience, data replication techniques are foremost means in such systems. Indeed, in many P2P applications, e.g. in a groupware, documents generated along application life-cycle can change over time. The need is then to efficiently replicate dynamic documents and data to support group processes and collaboration. In this paper, we propose a replication system for documents structured as XML files and evaluate it under different scenarios. The proposed system has a super-peer architecture that provides fast consistency for late joining peers. It uses optimistic replication techniques with propagating update operations from source to destination node in push mode. The system is suitable for asynchronous collaboration in online collaborative teams accomplishing a common project in a P2P environment.

KEY WORDS: Computational process, data replication; P2P; JXTA; peergroup architecture

1. INTRODUCTION

P2P computing system have emerged as large scale distributed computing paradigms. There is a lot of research conducted in such domain aiming to reveal the benefits and limitations of such

*Correspondence to: LSI/Technical University of Catalonia, C/Jordi Girona 1-3, 08034 Barcelona, Spain

system. In fact P2P systems are seen as digital ecosystems [7, 8] given their support to multi-disciplinary and multi-paradigmatic applications and having properties of self-organization inspired by natural ecosystems. Important features of such applications include the security, capability to be self-organized, decentralized, scalable and sustainable (see e.g. [1, 2, 10, 20]. P2P computing systems offer many advantages of decentralized distributed systems but suffer from availability and reliability. In order to increase availability and reliability, data replication techniques are considered commonplace in distributed computing systems [3, 21, 9, 16].

Replication can support complex computational processes in P2P systems, beyond simple file sharing, by increasing availability and reliability, and therefore, improve the perception of peers, yielding to fast response times and rich experience in such systems. Indeed, in P2P systems, users have often the perception of a highly unreliable system. The objective is then to increase the robustness of P2P applications. While this issue can be tackled from different perspectives, one interesting one is that of observing variations in the system, transform it into information that can support, in this case, collaboration among peers in a peergroup. In this sense, replication can be seen as a computational process, which by observing changes in data at some peer, generates events that are propagated to the other peers in the system to make the overall data consistent in a transparent manner to the peers. The replication as computational process can then be seen as distributed among different involved peers: changes are caused at some peer and require the involvement of a set of peers to bring the system in a consistent state.

Initial research work and development in P2P systems considered data replication techniques as a means to ensure availability of static information (typically files) in P2P systems under highly dynamic nature of computing nodes in P2P systems. For instance, in P2P systems for music file sharing, the replication allows to find the desired file at several peers as well as to enable a faster download.

However, considering only static information and data might be insufficient for certain types of applications. In many P2P systems the files or documents are considered static or, if the change, new versions are uploaded at different peers. In a broader sense, however, the documents could change over time and thus the issues of availability, consistency and scalability arise in P2P systems. Consider for instance, a group of peers that collaborate together in a project. They share documents among them, and, in order to increase availability, they decide to replicate the documents. Because documents can be changed by peers, for instance different peers can edit the same document, and thus changes should be propagated and made to the replicas to ensure consistency. Moreover, the consistency should be addressed under the dynamics of the P2P systems, which implies that some updates could take place later (as a peer might be off at the time when document changes occurred).

Replication can be seen as a family of techniques. Fundamental to any of them is the degree of replication (full vs. partial), as well as the source of the updates and the way updates are propagated in the system. It is therefore interesting to study different replication techniques for dynamic files to see their performance in the context of a peer-group. By increasing availability, fault-tolerance and response time, the replication helps improving the peers' perception of a robust and reliable system and improves user perceived *QoS* of the system. During the replication process, the information is gathered locally at peers' side but aims to keep globally consistent information at all peers.

In super-peer P2P networks, formed of different peer-groups, the replication can be handled at intra-group level and inter-group level. In the former, the content is replicated only within the peers of a peer-group, while in the later, replication of content can be done across different peer-groups. The intra-group replication, considered in this work, supports those groupware scenarios when disjoint groups of peers work separately on different projects and thus use group internal resources.

The rest of the paper is organized as follows. We analyse the existing replication techniques in Section 2 and propose in Section 3 a replication system for documents structured as XML files, with a super-peer architecture that provides fast consistency for late joining peers. The proposed replication system uses optimistic replication techniques with propagation update operations from source node to destination node in push mode. The system is suitable for asynchronous collaboration and works for both full and partial replication. The experimental study can be found in Section 4 and we conclude the paper with some remarks and directions for future work in Section 5.

2. INTRA-GROUP DATA REPLICATION TECHNIQUES IN P2P SYSTEMS

Data replication means storing copies of the same data at multiple peers thus improving availability and scalability. Full documents (or just chunks) can be replicated. Since the same data can be found at multiple peers, availability is assured in case of peer failure. Moreover, the throughput of the system is not affected in case of a scale-out as the operations with the same data are distributed across multiple peers. However, consistency is a challenge in replication systems that allow dynamic updates of replicas.

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 how replicas are distributed over the network (full vs. partial replication). We briefly describe them next.

2.1. 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.

The multi-master model allows multiple sites the right to modify their saved replica (Fig. 1). The system is responsible with propagating updates operated by a member of the group to

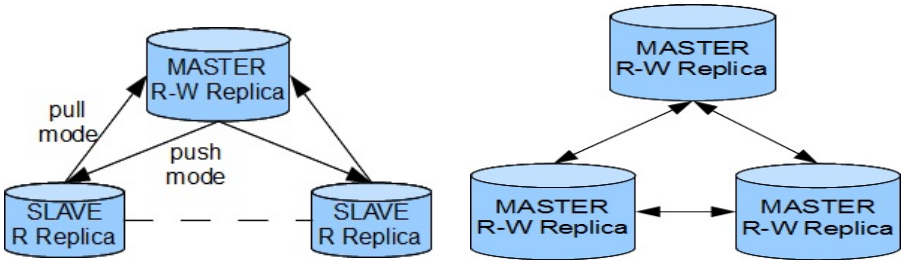

Figure 1. Single-master replication (left) and Multi-master replication (right).

all the other members and to solve any conflicts that may appear between concurrent changes made by different members [15]. 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.

2.2. Synchronous vs. Asynchronous Replication

These replication models use the notions of transaction, commit and abort. A transaction is a set of update operations (writes). 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. 2). 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 behaviour: two-phase-locking (2PL) [4], timestamp based algorithms, two-phase-commit protocol (2PC) [4].

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 [12] proposed a new protocol that makes eager replication avoid the drawbacks of the above protocols. The protocol takes advantage of the rich semantics of group communication primitives and the relaxed isolation guarantees provided by most databases.

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 after that the

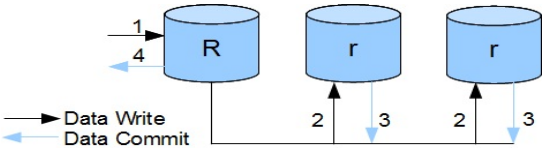

Figure 2. Synchronous replication.

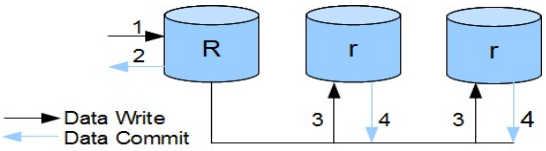

Figure 3. Asynchronous replication.

updates are propagated to the remote sites (Fig. 3). The asynchronous replication technique can be classified as optimistic or pessimistic in terms of conflicting updates. The optimistic approach assumes that conflicting updates will take place rarely [15]. Updates are propagated in the background and conflicts are fixed after they happen. On the other hand, the pessimistic approach assumes that conflict updates are likely to occur and implements mechanisms for replica consistency [18, 6].

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.

2.3. 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 space and reduces the number of messages needed to update replicas since updates are propagated only to the sites that hold a certain replica. Thus, updates produce reduce load 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 [19]. In partial replication, it is important to find a right replication factor.

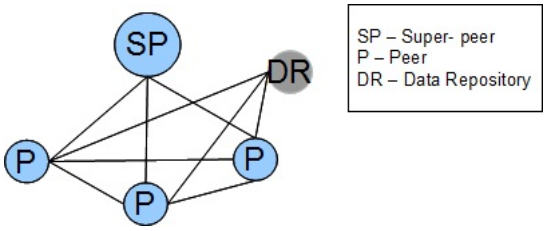

Figure 4. Peer-group structure with a Data Repository.

Careful planning should be done when deciding which documents to replicate and at which peers.

3. SYSTEM ARCHITECTURE

This section presents the architecture of a replication system for XML files with a super-peer architecture that provides fast consistency for late joining peers. 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.

3.1. Peer-group Design

A peer-group is a gathering of multiple peers who can be geographically far away from one another but who have a common goal. For example, they could all work together accomplishing a project.

The peer-group is organized in such a manner that each peer can directly communicate with any other peer in the group. Furthermore, the peer-group has a central manager, the super-peer, who 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. 4). When a peer receives a task, it then connects to the DR to get copies of the documents which it can modify afterwards.

We use JXTA library [5, 13, 17, 11] 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

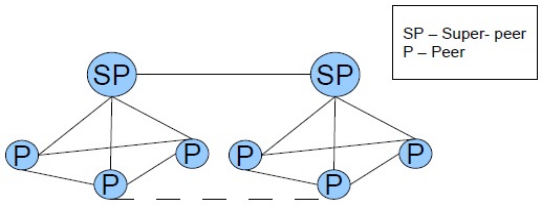

Figure 5. Super-peer group structure.

group. The communication within the members of the group is done using unicast and multicast communication.

3.2. Peer-group Entities

3.2.1. 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 advantages of having a super-peer is that job submissions and data queries arrive faster to the destination. The super-peer in our peer-group structure plays the role of a rendezvous peer: 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.

3.2.2. Peers

Peers are responsible for solving 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 behaviour. One version uses multicast in order to propagate the changes to members of the group while the other version uses direct communication.

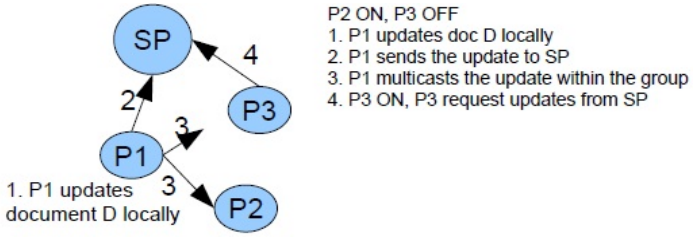

Figure 6. Multicast Propagation Algorithm.

3.2.3. 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.

3.3. Peer Propagation Behaviour

3.3.1. 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.

Multicast propagation algorithm is depicted in Fig. 7. Special cases must be considered when using multicast propagation. P2 might not receive the update (packet lost). In this case, P1 must retransmit the packet or P2 polls the super-peer periodically for newer versions of document D and retrieve from super-peer new versions, if any. Another problem is that P2 might receive unordered updates as multicast doesn't guarantee order delivery. P2 can deal with it by buffering unordered updates and order them locally or can request missing updates from the super-peer. For simplicity, we use the interaction of the peer with the super-peer in both situations. Peers will drop duplicate updates.

3.3.2. 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

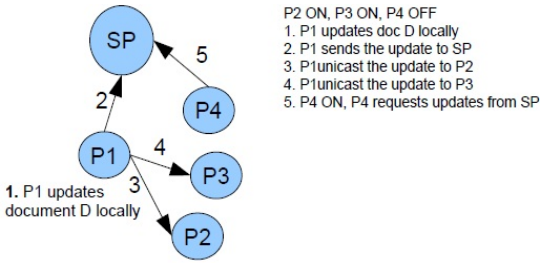

Figure 7. Unicast Propagation Algorithm.

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. An example of unicast propagation algorithm is depicted in Fig. 7. Special cases must be taken into account. P1 disconnects after updates were sent to SP and P2, but before they are sent to P3. A solution to this problem is for P1 to send the updates not received to P3 after it rejoins the group. This means that P1 must store in its cache unsent updates for each peer. The second solution is for P3 to poll periodically the super-peer for newer versions of the document and retrieve them from SP if they exist.

3.4. Interactions Between Entities

The peer communicates 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.

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

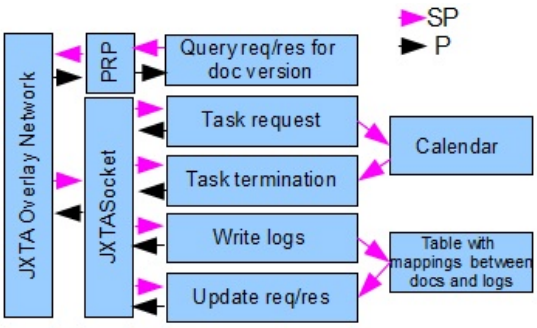

Figure 8. Peer-Superpeer Interaction.

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 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 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 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.

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

Peer - Data Repository Interaction. The peer uses the Content Management Service offered by JXTA to download files from a specified location [11].

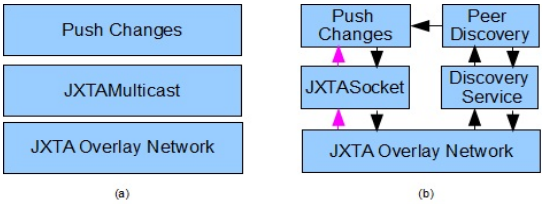

Figure 9. Peer-to-Peer Interaction.

4. EXPERIMENTAL EVALUATION

Our goal is to determine the consistency factors and analyse their influence over the consistency time of our proposed replicated system. For the purposes of the study we used nodes of a cluster environment. We first study the propagation time in the network and then evaluate how does the structure of the write log affect the performance of enforcing received updates at client side. Moreover, we will analyse the super-peer cache load.

The average time for a document to achieve consistency is a sum between the time needed to propagate the updates (network delivery time) and the time needed to operate the changes in the local database (client update time), averaged over the number of documents. The network delivery time depends on several factors like the overlay network and overlay network topology, the number of peers in the peer group, the size of the write log that is propagated within the group, the number of write logs that each peer is propagating, etc.

4.1. Network Delivery Time

In order to evaluate the network delivery time for our replicated system we simulated different scenarios for the asynchronous collaboration within groups with one super-peer and 6, 9 or 12 peers. The scenarios differ by the number of peers that perform local updates. A certain document is supposed to be updated by a single peer, which is responsible to propagate the updates (write log) made to the document to the rest of the peers. We have performed tests for both multicast model and unicast model. Multicast limits the maximum size of a datagram (64KB). Thus, in the cases where the write log exceeds the maximum allowed we split the write log into multiple datagrams. In out tests, we use datagrams of size 12KB. In the case of unicast model, we don't split any write log. We send it fully through a socket channel.

First scenario. In the first scenario, we assume that only one peer has the right to update document replicas and propagate updates within the group.

Table I shows the average time needed for logs of sizes 12KB, 1.2MB and 120MB to be propagated within groups of 6, 9 and 12 peers using IP multicast propagation. It can be seen from the table that all the peers in the group receive the updates almost in the same time. This happens due to the cluster's structure - the nodes of the cluster are linked just to one

Table I. Average Time (ms) for Multicast Propagation (Scenario 1)

Log size	**No. of datagrams (size 12KB)**	**No. of peers**		
		6	**9**	**12**
12KB	1	6	6	6
1.2MB	100	220	218	230
120MB	10000	270	275	273

Table II. Average Time (ms) for Unicast Propagation (Scenario 1)

Log size	**No. of peers**		
	6	**9**	**12**
12KB	201	317	412
1.2MB	423	542	636
120MB	2132	3271	3955

switch, and because the flow is unidirectional - just one node multicasts, the others receive. But, the propagation time depends on the log size. A greater log size means a higher number of datagrams to be transmitted. No packets were dropped off.

Table II shows the total time needed to propagate an update to all peers using unicast method. The delivery time increases with the number of peers, as more unicast transmissions need to be done. Also, the delivery time increases with increasing the size of the write log. Thus, in the case where just one peer pushes updates within the group, multicast propagation offers best results in time delivery.

Second scenario. In the second scenario, all peers have rights to update document replicas. In this scenario we also have groups of 6, 9 and 12 peers. In the case of multicast, each peer that propagates logs splits them in datagrams of 12KB whenever the log size exceeds 12KB. Every peer must push one update (one write log) and receive updates from all other peers. Table III shows the average time for multicast propagation for the second scenario. With peers multicasting larger size logs, the average propagation time increases significantly, as the links are flooded with datagrams. Moreover, in these cases some packets are also dropped. In the case where each peer multicasts a log of size 1.2MB there is a drop packet rate of 10% and in the case where each peer multicasts a log of size 120MB the drop packet rate is of 50% . As the average multicast speed is 600MB, the links were flooded with multicast packets when 120MB logs were multicasted. This led to a significant decrease in transfer speed.

Table IV shows the total propagation time in the case of unicast propagation for the second scenario. The results show that, with increasing the number of peers, the total propagation time increases significantly. This is due to the fact that now the connection link is shared for both uploading and downloading. Another important fact is that unicast performs much better when propagating large logs. The overall time to propagate through unicast the log of size 120MB to all the peers in the group is less than the time needed to multicast the same log.

Table III. Average Time (ms) for Multicast Propagation (Scenario 2)

Log size	**No. of datagrams (size 12KB)**	**No. of peers**		
		6	**9**	**12**
12KB	1	6	6	6
1.2MB	100	289	293	302
120MB	10000	18345	18403	18728

Table IV. Average Time (ms) for Unicast Propagation (Scenario 2)

Log size	**No. of peers**		
	6	**9**	**12**
12KB	638	827	1124
1.2MB	1372	1923	2479
120MB	5413	6202	7413

Table V. Average Time (ms) for updating local database.

	Size of write log (bytes)	
No. of operations	**600**	**1200**
1	3	4
100	466	517
200	671	683

A reason for this is that, with multicasting switches and routers must take care to replicate the datagrams, which takes time, and then to route them to destination.

4.2. Client Update Time

The time needed to operate the changes in the local XML database does not depend on the size of the write log, but on the number of operations in the write log. Table V shows the time in milliseconds needed to perform updates on the local database taking into account the number of operations and the size of the write log. It can be seen that executing just one operation that updates one field with a new content of size 600 bytes takes very little time comparing to executing 100 operations for updating fields with new content of size 6 bytes each or to executing 200 operations for updating fields with new content of size 3 bytes each.

4.3. Late Join Data Delivery

When a peer joins the network, the first thing it does is to update its current state. Thus the data transmission is initiated by the peer itself when sends an update message to the super-

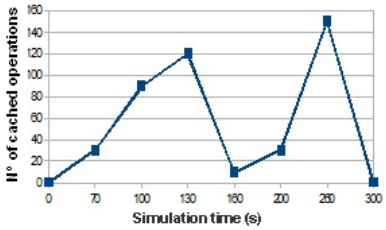

Figure 10. Number of operations cached at super-peer.

peer. We chose this solution in order not to cache operations at peers sites and thus polling each peer in the group which would have caused too much traffic and the network could have got saturated. Another advantage is that the newly joined peer obtains the updates fast, as it only has to contact a single node (the super-peer) which already has all the updates.

The update message encapsulates the IDs of the documents for which the peer wants to retrieve updates and the current versions of the documents. The super-peer is responsible with delivering the updates the peer requested as soon as possible. For this reason, the super peer uses a hash table which maps document IDs with a vector of write logs.

The write logs for a certain document ID are inserted into a vector which is sorted by the version of the update. The super-peer then retrieves from its cache with a complexity of $O(1)$ the updates to be transmitted to the peer.

4.4. Super-peer Cache

In our super-peer network, the super-peer is responsible with caching all the updates operated by all the peers in the group. Thus, the super-peer requires a rather large storage capacity. But even in this case, the upper limit can be reached. It is therefore necessary for old cached updates to be dropped.

As each peer polls periodically the SP for new updates, the SP keeps track of which updates were operated at every peer. Each online peer propagates an update operation of size 12KB every 1 sec and polls the SP for new updates when it joins the group and after at intervals of 30 secs. Fig. 10 shows the number of operations cached at SP. It can be noticed that SP caches more updates when some peers of the group are offline and caches less updates when all peers are online. In the case where all peers are online, the SP empties its cache when all peers send requests for new versions in the same time.

4.5. Evaluation

Multicast propagation proved efficient when there is just one source node within the group and logs have a reduced size. On the other hand, unicast propagation proved efficient when

there are multiple sources that push updates in the network and also when the size of the logs is rather large.

The consistency time of the system depends on several factors: the underlying network topology, the size and structure of the write log, the number of peers that have updates to propagate, the number of peers in the group, the number of documents. The experimental results show that groups with peers that propagate large sized logs need more time to get into a consistent time than groups where peers propagate short logs. On the other hand, a great number of peers transmitting through multicast will flood the links and thus datagram packets are dropped and updates lost. The structure of the write log will affect the time a client needs to enforce the updates in its local database and thus influencing the consistency time of the system. If the logs contain many operations then the time needed to update the XML file is greater than in the case the log contains less operations. The more documents a peer has, the more time it takes to propagate updates or to enforce received updates.

5. CONCLUSIONS AND FUTURE WORK

P2P computing systems can leverage large amount of computational resources and can support in many ways computational processes in complex applications in a distributed, decentralized, scalable and secure manner. For example, P2P systems can support group collaboration among peers of a peergroup. One weak point in such system is the lack of high content availability, mainly due to dynamics of such systems and lack of central entities. This often yields a poor perception of peers about the potential of P2P systems. In order to increase availability and reliability, and therefore, improve the perception of peers, yielding to fast response times and rich experience, data replication techniques are are foremost means in such systems.

We have designed a replication system for XML files, having a super-peer architecture that provides fast consistency for late joining peers. The replication system uses optimistic replication techniques with propagating update operations in push mode. Our system permits both full and partial replication. We analysed both unicast and multicast propagation. Though multicast ensures a fast delivery of content to all members of the group, the datagram packet is limited in size. Thus, write logs exceeding this size must be split up and mechanisms for packet ordering at the destination must be used. With multicast, packets might be dropped or lost. That is why this system is suitable for short logs. For logs of large sizes, the unicast version of the system is necessary. For a further study, it is necessary to analyse other consistency factors like scalability of the replication system in terms of number of documents per peer, the performance regarding the structure of the XML tree. It is also important in the case of partial replication to find experimentally empirical values for replication factors.

In the future, we plan to implement a version of this replication protocol using Application Layer Multicast Protocol based on Scribe [14] which ensures reliable transmission of data through multicast trees having unicast connections.

REFERENCES

1. J. Arnedo, K. Matsuo, L. Barolli, F. Xhafa (2011). Secure Communication Setup for a P2P based JXTA-Overlay Platform. *IEEE Transactions on Industrial Electronics*. Volume: 58 Issue: 6, 2086-2096.
2. L. Barolli, F. Xhafa (2011). JXTA-Overlay: A P2P Platform for Distributed, Collaborative, and Ubiquitous Computing. *IEEE Transactions on Industrial Electronics*. Volume: 58 Issue: 6, 2163 - 2172
3. U. Bartlang and J.P. Muller (2010). A flexible content repository to enable a peer-to-peer-based wiki. *Concurrency and Computation: Practice and Experience*, 22: 831-871.
4. Ph. Bernstein, V. Hadzilacos, N. Goodman. *Concurrency Control and Recovery in Database Systems*. Addison-Wesley 1987, ISBN 0-201-10715-5, 1987
5. D. Brookshier, D. Govoni, N. Krishnan, and J. Soto. *JXTA: Java P2P Programming*. (Sams Pub., 2002)
6. C. Coulon, E. Pacitti, P. 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)*, p.809-815, July 20-22, 2005
7. H. Dong, F. Khadeer Hussain, E. Chang. A Human-Centered Semantic Service Platform for the Digital Ecosystems Environment. World Wide Web 13(1-2): 75-103, 2010.
8. H. Dong, F. Khadeer Hussain, E. Chang. A context-aware semantic similarity model for ontology environments. Concurrency and Computation: Practice and Experience 23(5): 505-524, 2011.
9. A. Elghirani, R. Subrata and A.Y. Zomaya (2009), Intelligent scheduling and replication: a synergistic approach. *Concurrency and Computation: Practice and Experience*, 21: 357-376.
10. T. Enokido, A. Aikebaier, M. Takizawa. Process Allocation Algorithms for Saving Power Consumption in Peer-to-Peer Systems *IEEE Transactions on Industrial Electronics*. Volume: 58 Issue: 6, 2097 - 2105
11. JXTA Java Standard Edition v2.6: Programmers Guide, 2010
12. B. Kemme, G. Alonso. A new approach to developing and implementing eager database replication protocols. *ACM Transactions on Database Systems*, 25(3): 333-379, 2000
13. S. Li. *Early Adopter JXTA* (Wrox Press, 2003).
14. J. Ludwig, B. Israel. Multicast Streaming with SplitStream. TechRep, Rochester Institute of Technology, USA, 2004
15. V. Martins, E. Pacitti, Patrick Valduriez. Survey of data replication in P2P systems, *Tech Report* NR-6083, INRIA, France, 2007
16. C. Nicholson, D.G. Cameron, A.T. Doyle, A.P. Millar and K. Stockinger. (2008), Dynamic data replication in LCG 2008. *Concurrency and Computation: Practice and Experience*, 20: 1259-1271.
17. S. Oaks, B. Traversat, and L. Gong. *JXTA in a Nutshell* (O'Reilly, 2003).
18. E. Pacitti, P. Minet, and E. Simon. 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)*, San Francisco, CA, USA, 126-137, 1999.
19. Y. Saito and M. Shapiro. 2005. Optimistic replication. *ACM Comput. Surv.* 37, 1 (March 2005), 42-81
20. A.B. Waluyo, W. Rahayu, D. Taniar, B. Scrinivasan (2011) A Novel Structure and Access Mechanism for Mobile Data Broadcast in Digital Ecosystems. *IEEE Transactions on Industrial Electronics*. Volume: 58 Issue: 6, 2173 - 2182
21. J. Zhang and P. Honeyman (2008),. A replicated file system for Grid computing. *Concurrency and Computation: Practice and Experience*, 20: 1113-1130.