# A Demand based Algorithm for Rapid Updating of Replicas

Jesús Acosta-Elias, Leandro Navarro-Moldes
Polytechnic University of Catalonia, Spain
{jacosta, leandro}@ac.upc.es

## Abstract

*In many Internet scale replicated system, not all replicas can be dealt with in the same way, since some will be in greater demand than others. In the case of weak consistency algorithms, we have observed that updating first replicas having most demand, a greater number of clients would gain access to updated content in a shorter period of time.*

*In this work we have investigated the benefits that can be obtained by prioritizing replicas with greater demand, and considerable improvements have been achieved. In zones of higher demand, the consistent state is reached up to six times quicker than with a normal weak consistency algorithm, without incurring the additional costs of the strong consistency.*

Keywords: Weak consistency, consistency algorithms, replication, distributed system.

## 1. Introduction[1]

There is a growing interest on Internet scale distributed systems where many potential clients may contact a single host to request a given service at almost the same time from several locations. The presence of replica servers may help to improve the situation because clients will be able to contact the nearest replica. A Replica is a host who provides exactly the same services as the principal host. In this paper we will use the terms server and replica in the same sense.

Content replication between servers in a distributed system is justified by the need to reduce delay, to provide availability and to be scalable [11], to tolerate failure in the links, and also to withstand segmentation. The algorithms currently available for replica updating can be broadly classified into two groups, according to their consistency:

- Strong consistency, and
- Weak consistency

Strong consistency algorithms are costly, non-scalable on networks, not very reliable, generate considerable latency and a great deal of traffic. They are suitable for systems with a small number of replicas, in which it must be guaranteed that all the replicas are in a consistent state (i.e. all the replicas possess exactly the same content) before any transaction can be carried out (synchronous systems) [3, 14].

However, weak consistency algorithms [7, 13, 1] generate very little traffic, low latency, and are more scalable. They do not sacrifice either availability or reply time in order to guarantee strong consistency, but only need to ensure that the replicas eventually converge to a consistent state in a finite, but not bounded, period of time. They are very useful in systems where it is necessary for all the replicas to be totally consistent in order for transactions to be carried out (systems that withstand a certain degree of asynchrony). This is the case of Usenet news, or in computer-supported cooperative work systems.

With the weak consistency algorithm [7], each server (replica) from time to time chooses a neighbour to start an update session. In an update session two servers mutually exchange summary vectors then they exchange some data to mutually update their contents. At the end of the session both servers will have the same mutually consistent content. These are called anti-entropy sessions: It is called an anti-entropy session because in each session between replicas, the total entropy in the system is reduced. In this paper it will be referred to simply as a “session”.

The metric principle to be employed is how many sessions are necessary for a change brought about in a replica to be propagated to all the others.

Golding [7] demonstrated that the neighbouring server’s random choice has the best performance (fewest number of sessions) for maintaining the consistency of the replicas in a peer-to-peer network. This gives rise to the fact that all the replicas are updated, no matter how much demand (number of requests per unit of time) they have, in such a way that a replica with low demand can be updated first, before another with much greater demand.

In most distributed applications, some replicas tend to have more demand than others due to different factors, such as:

- Geographical distribution
- Number of clients,
- Number of requests arising from more intense work among clients.

[1] This work has been partially supported by the Mexican Ministry of Education (Secretaría de Educación Pública) under contract PROMEP-57, and Spanish MCyT project COSACO.

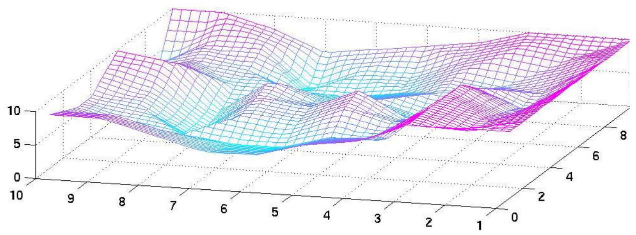

Fig. 1. Example of demand in a large distributed system

Thus if we draw a graph of the distribution of the replicas in the X-Y plane, and their demand along the Z axis, we will have an image of hills and valleys in which the valleys, in our case, are the areas of greater demand and the replicas with least demand are on the hills (see Fig. 1).

This is analogous to the relativistic effect where mass causes space to curve, therefore objects in that curved space will have their paths attracted by massive objects. In our case, demand is analogous to mass. Therefore, updates are attracted or directed to nodes or regions with higher demand.

An algorithm giving priority to the delivery of updates to replicas with higher demand will enable to satisfy requests to more clients with up-to-date content in less time.

That is why replica demand must be taken into account when designing algorithms for maintaining consistency of replicated data.

The rest of the paper is organized as follows. Section 2 describes how our algorithm directs the propagation of updates to nodes with greater demand. Sections 3 and 4 describe the dynamic situation when demand changes at the same time as updates are being propagated. Section V describes how this algorithm has been validated by simulation on several simple and complex topologies, including random topologies with topologic properties equivalent to the real Internet. The results indicate that the number of sessions to reach a global consistent state may be related to the diameter of the network instead of the number of nodes. Section 6 describes ongoing work to improve the efficiency of this protocol on topologies with several greater demand regions surrounded by low demand regions. Section 7 describes related work, and section 8 gives some conclusions.

## 2. Proposition

The model of our distributed system consists of a number of nodes, N, that communicate via message passing. We assume a fully replicated system, i.e., all nodes must have exactly the same content. Every node is a server that gives services to local clients. Clients make requests to a server, and every service request is a *"read"* operation, a *"write"* operation, or both. When a client invokes a *"write"* operation in a server, this operation (change) must be propagated to all servers (replicas) in order to guarantee the consistency of the replicas. An *update* is a message that carries a *"write"* operation to replica in other neighbouring nodes. In this model, the *demand* of a server is measured as the number of service requests by their clients per time unit. From our model we are interested in an algorithm for consistency giving priority to nodes with high demand.

Our proposition consists in all the servers selecting, at random time, the neighbour with whom they will begin a consistency session. This choice must not be made at random order as in [7], but rather the neighbour with most demand must be chosen first. If the neighbour selected has another neighbour with even greater demand the process will be repeated, which will cause the replica updates to literally flood the valleys (zones of greater demand).

In order to describe this algorithm, we take a segment and locate within it five replicas that are on a slope, and which are suitable for this purpose.

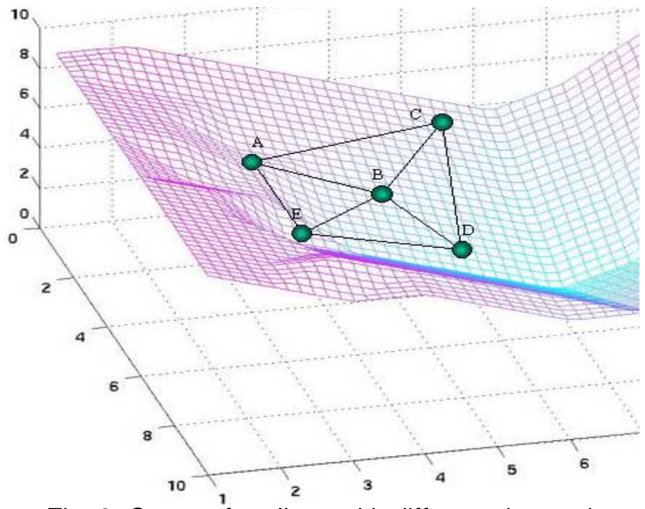

Fig. 2. Group of replicas with different demand

In Fig. 2 we have a group of replicas in which the position on the Z axis corresponds to their demand. As Z tends to zero, demand increases, in such a way that the replica at the lowest part of the slope, or area of least energy, is the replica with greater demand.

The following table shows a list of these replicas with their corresponding demand, expressed by the number of requests per unit of time:

| Replica | A | B | C | D | E |
|---|---|---|---|---|---|
| Rate of demand (Z axis) | 4 | 6 | 3 | 8 | 7 |

In the baseline weak consistency algorithm, at random time, every node will execute the same algorithm. Let's take B as an example: in order to initiate the updating process replica B selects a random neighbour (say B-C, which means that replica B selects replica C to initiate an anti-entropy session). Several things may occur: in the following we provide examples of two extreme cases:

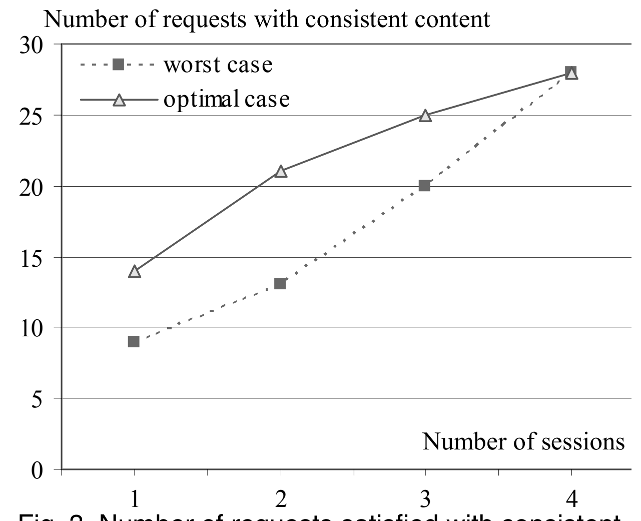

Fig. 3. Number of requests satisfied with consistent content as time goes on (number of sessions or time units)

1) The worst case (B-C, B-A, B-E, B-D): In time 1 replicas B and C finish an update session; their content is now mutually consistent, so when this period of time comes to an end both replicas serve a total of nine (B:6+C:3) requests satisfied with updated content (Fig.3); next time B-A, then B-E, and finally B-D.
2) The best case (B-D, B-E, B-A, B-C): In time 1 replicas B and D are able to serve fourteen requests satisfied with updated content (Fig.3).

Furthermore, in the same time 1 replica D is able to distribute updated content to adjacent replicas in the lowest part of the valley, should these exist.

While the random weak consistency algorithm in [7] gives random performance ranging between the worst and optimal case in Fig. 3, our algorithm works even better than the optimal case above. This is a consequence of two optimizations in our algorithm: (1) neighbours are elected orderly by demand instead of random order, and (2) messages are immediately propagated to the neighbour with highest demand instead of waiting for the next session, a random wait.

### 2.1 Proposed algorithm.

Our proposed algorithm (called fast consistency), described in terms of the example illustrated by fig. 2 and fig. 3, is as follows:

Letters E, B and D represent the replicas. The algorithm starts after E has experienced a write operation.

This algorithm has two parts. The first part, which is the weak consistency part that permits to bring updates to all replicas, where neighbours are selected in order of demand at each node. The second part, permits a fast update of replicas by prioritizing the servers with higher demand.

/* weak consistency */

1. After random time the replica E select the neighbour B (most demand).
2. E Sends to B a message to request for initiate a session.
3. B receives the request from E.
4. B sends to E its summary vector.
5. E receives the summary vector from B.
6. E sends its summary vector to B.
7. Replica E determines if it has messages that B has not yet received, by seeing if some of its summary timestamps are greater than the corresponding ones its partner(replica B).
8. E sends to B the messages that B has not seen before.
9. B receives the summary vector from E.
10. Replica B determines if it has messages that E has not yet received, by seeing if some of its summary timestamp are greater than the corresponding ones its partner (replica E).
11. B sends to E the messages that E has not received.
12. Replica B receives a new message from replica E (as a result of step 8).

/* fast update: occurs as a result of a new update message, either coming from a client, or from an anti-entropy session */

13. Immediately B sends to replica D a request for fast update. This request has information (id and timestamp) of new arrived messages to replica B.

Note that in fast update sessions the summary vectors are not exchanged.

14. Replica D receives the request of fast update.
15. If D does not have the messages, answer with YES. Else answer with NO.
16. B receives a message from D.
17. If the answer of D is YES, B sends the message with that update.
18. If the answer of D is NO, B sends nothing.

## 3. Dynamic model

In the previous model, it is assumed that the request demand conditions do not change with time, but what happens if in fact these conditions do change with time? In that case, the previous algorithm does not work.

Let us see why:

In Figure 4 we have four replicas, in time 1 replica B has a table where we find its neighbours with their respective demands, arranged according to these demands:

| Replica | Requests |
|---|---|
| D | 13 |
| A | 2 |
| C | 0 |

Replication process:

- Time 1 - B starts the update process with the replica with greatest demand (D).

- Time 2 - B starts the process with the following replica (A), which would correspond to it according to the table of neighbours.

But for this time 2 in replica A, the amount of requests changed – falling from 2 to 0 (A' in Fig.4), and replica C changed from 0 to 9 (C' in Fig.4), so that if B followed the static algorithm it would not contribute to carrying consistency to the zones with greatest demand.

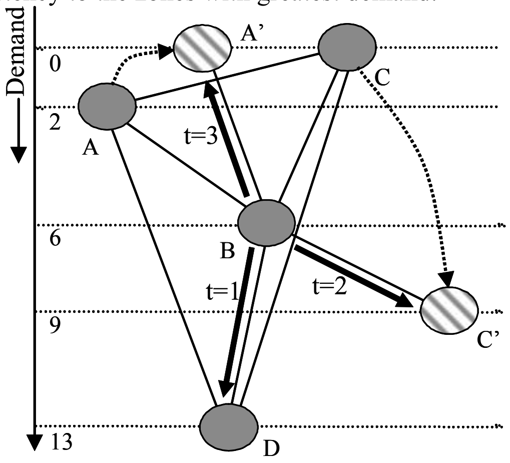

Fig.4. Requests at the nodes, and sessions

## 4. The dynamic algorithm

Each replica maintains a table with its neighbours' data. The table holds at least an identifying name and its demand (requests per unit of time). Before any replication process is carried out, this table must be updated. This updating provides us with knowledge pertaining to the replica appropriate for carrying out content updating, depending on the demand, and, as an added advantage, tells us if this replica is available (link and server both working).

Example:

We draw up a table with the replicas from Fig.4, with B neighbours' vector arranged in decreasing order of demand.

| Time | 1 | 2 | 3 |
|---|---|---|---|
| Sessions | B-D | B-C' | B-A' |
| B version vector | A | A' | |
| | C | | |

- At time=1, replica B starts update with replica D (at that moment the one with greatest demand).
- At time=2, replica C has greatest demand, and we symbolize it as C' (Fig.4). If B knows about this, B starts a session with replica C'.
- At time=3, only replica A' remains, and with this replica content update is carried out.

We assume that every node is periodically informed of the demand of their neighbours, in a way similar to IP routing algorithms.

## 5. Validation

To validate this proposition a simulator has been built using the NS [13] network simulator as a base, and the necessary agents and scripts are written on ns.

For the choice of simulation conditions, we set as our aim that these conditions be representative of Internet [5, 2], not forgetting how difficult it can be to take into account the characteristics of Internet for a simulation [6]. It is for that reason we have chosen BRITE [10] to generate the topologies used in our simulations, since this software generates topologies that fulfil Internet power laws [9, 5].

Faloutsos et al. in [5] describe the following power law relationships (power laws are equations of the form $y = x^a$): outdegree of node versus rank, number of nodes versus outdegree, number of node pairs within a neighborhood versus neighborhood size (in hops), and eigen values of the adjacency matrix versus rank. Medina et al. in [9] suggest two main factors in the formation of Internet topologies: (F1) preferential connectivity and (F2) incremental growth. F1 dictate the tendency of a new node to connect to those existing nodes that have higher outdegrees. F2 dictates that new nodes join the Internet in an incremental way.

With BRITE we generate the random topologies, assigning to each replica, also in a random way, their respective demands. The simulation begins by assuming a change on a randomly chosen replica, with the aim of measuring the number of sessions the algorithm uses to propagate this change, both in the replica with most demand and in those with less demand. Simulations were carried out with 50 and 100 replicas, and experiments were repeated 10,000 times. The results can be seen in figures 5 and 6.

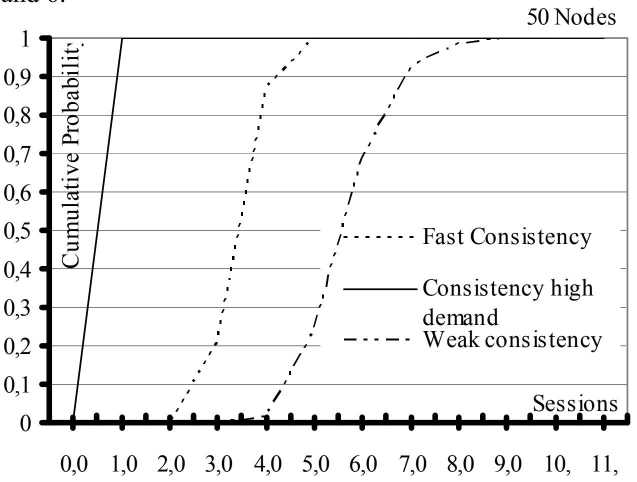

Fig. 5. CDF of number of sessions for 50 nodes

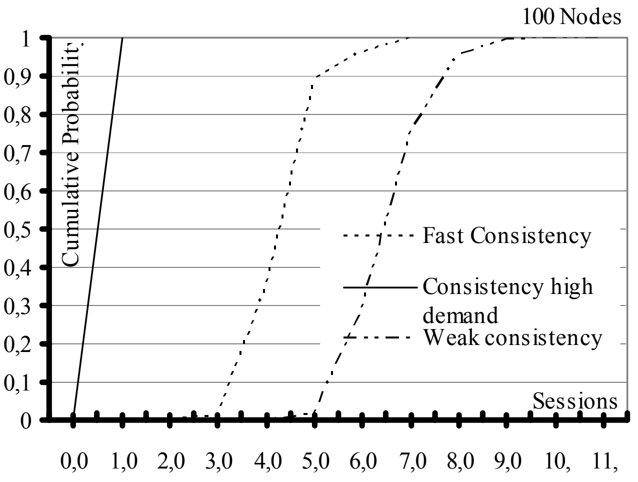

Fig. 6. CDF of number of sessions for 100 nodes

In both cases, with 50 or 100 replicas, we observe that the change arrives quickly to the replicas with most demand (unbroken line), causing these replicas to reach the state of consistency on an average of 1 session, in the case of 50 replicas, and also an average of 1 session in the case of 100 replicas. On the other hand, the change takes on average 3.9261 sessions to reach all replicas, in the case of the 50 servers, and 4.78117 sessions in the case of the 100 servers.

The time it takes for the message to arrive to the server at the lowest point in the valley is in fact the propagation delay associated to the link, although for purposes of comparison in Figs. 5 and 6 it is expressed in average session times.

The change takes on average 6.1499 sessions to reach all the replicas in the 50 replica case, and 6.982 sessions in the 100 replica case, using the pure weak consistency algorithm (Figs. 5 and 6). As one may observe, our proposition not only substantially improves the areas of most demand, but also improves it in general for all the replicas.

Similar results as shown in figures 5 and 6 have been obtained with simpler uniform topologies (linear, ring, grid), with different number of nodes. In figure 6 can be observed that as the number of nodes doubles, the number of sessions required to propagate a change to all replicas does not grow as fast. It seems that the number of sessions required to reach a global consistent state is related to the diameter of the network.

As the mean number of sessions to reach a global consistent state is related to the diameter of the network and it does not change significantly with the number of nodes, the result seems to be applicable to the whole Internet with a huge number of hosts but a diameter [2] in the order of 20.

## 6. Complex demand distribution

As a consequence of the faster update of replicas with higher demand, in the longer term those replicas with lower or reduced demand will tend to have less updated (i.e. stale) content. This can lead to the appearance of clusters of highly consistent replicas (islands), surrounded by regions with less consistent content.

Work under way is investigating how these islands can be characterized, which mechanisms can help to interconnect them, for instance a leader election algorithm for each island, with leaders becoming part of an island interconnection network. This will help to ensure that all updates will reach very fast to any region with high demand, avoiding that regions of low or null demand would slow down the propagation of updates.

## 7. Related work

There are some proposals that set forth the necessity for the replicas to be adapted to customer requirements, so in this section we briefly describe some of the work dealing with this area of study.

Richard Lenz [8] proposes ASPECT (Application Oriented Specification of Consistency Terms), which enables us to specify weak consistency requirements from the point of view of the application according to the "need to know" principle, which states that "data only have to be made available where they are needed and only as current and consistent as required by the applications that access these data". It proposes the use of two dimensions to specify the replica consistency requirements, one spatial and one temporal; the spatial dimension describes the degree of consistency or quality of data of a particular copy. The temporal dimension specifies when this quality of data is required. As we may observe, this proposition is only a specification.

Petersen et al. in [13] describe the Bayou anti-entropy protocol, which facilitates the propagation of replicas by means of weak consistency. They pose, furthermore, the possibility of providing a variety of policies for where and when the updating may be propagated.

The question is also raised of furnishing four types of policy in the anti-entropy protocol: Policies for when to reconcile; policies for selecting with which replicas to reconcile, policies for deciding how aggressively to truncate the write-log, and policies for selecting a server from which to create new replicas.

- Policies for when to carry out reconciliation: Potential policies could be periodic reconciliation, manually initiated reconciliation, or those initiated by the system.

- Policies for selecting with which server the anti-entropy session can be started may depend on many factors: what other replicas are attainable, for example; characteristics of the network connections using these replicas.

- Policies for selecting how aggressively to truncate the write-log enable us to negotiate storage and network resources necessary in an anti-entropy session. Truncating the write-log very aggressively can give rise to very long anti-entropy sessions among some servers due to the need to transfer complete databases.

- When various servers are available for creating a new replica, quantities to be considered must be identified, in addition to other characteristics of these replicas, how out of time they are, band width of connections, and how complete their write-logs are.

This proposition enumerates some interesting policies. However, it is no more than this – an outline of possible policies, none of which have been researched in detail, neither from the point of view of implantation nor performance.

Brun-Cotan and Makpangou [4]: This article presents an architecture and a run-time for "adaptable replicated objects". Different applications require different contracts concerning the consistency of replicas, or a different negotiation between performance and consistency. The architecture proposed structures a replicated object into three component classes: access objects, replicas, and consistency managers. Together with the access object, the consistency manager are the components responsible for maintaining consistency in each object, depending on application needs; that is to say, the application for a particular activity may require strong consistency, and in other cases weak consistency.

This architecture is conceived for the design of distributed applications, and its advantage is the existence of a contract that specifies the consistency of the objects replicated, in which the degree of required consistency is defined, whether it be strong or weak.

## 8. Conclusions

This algorithm ensures rapid content update propagation towards replicas located in the zones of least energy (greatest demand). Those replicas having none –or very few– requests will also be updated, but at a normal speed, so that in a finite, but not bounded, time they will be consistent. However, replicas having the greatest number of requests will have contents that will be rapidly updated.

Our algorithm is quite simple; it requires few additional bytes in the exchange of messages between replicas, and thus a greater number of clients/customers/users can be satisfied in the first sessions. The algorithm is scalable, does not cause traffic overload, and also increases replication speed.

The worst case would be when all the replicas possess the same demand; in such a situation the algorithm behaves like a normal weak consistency algorithm.

## 9. References

[1] A. Adya, "Weak Consistency: A Generalized Theory and Optimistic Implementations for Distributed Transactions", *PhD thesis Massachusetts Institute of Technology*, Department of Electrical Engineering and Computer Science, March 1999.

[2] R. Albert, H. Jeong, and A.-L.Barabási, "Diameter of the World-Wide Web", *Nature*, page 130, September 1999

[3] K. P. Birman, "The process group approach to reliable distributed computing", *Communications of ACM*, Decembre 1993/Vol. 36, No. 12

[4] G. Brun-Cota, M. Makpangou, "Adaptable Replicated Objects in Distributed Enviroments", Research Report 2593, INRIA, May 1995.

[5] M. Faloutsos, P. Faloutsos, and C. Faloutsos, "On Power-Law Relationships of the Internet Topology", *ACM SIGCOMM*, Cambridge, MA, September 1999

[6] S. Floyd, and V. Paxson, "Dificulties in Simulating the Internet", *IEEE/ACM Transactions on Networking*, Vol 9,no. 4, August 2001.

[7] R. A. Golding, "Weak-Consistency Group Communication and Membership", *PhD thesis, University of California*, Santa Cruz, Computer and Information Sciences Technical Report UCSC-CRL-92-52, December 1992.

[8] R. Lenz, "Adaptive distributed data management with weak consistent replicated data", *ACM Symposium on Applied Computing*, February 1996

[9] A. Medina, I. Matta, and J. Byers, "On the Origin of Power Laws in Internet Topologies", *ACM Computer Communication Review*, pages 160-163, April 2000.

[10] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: Universal Topology Generation from a User's Perspective",

[11] B. C. Neuman, "Scale in Distributed Systems. In Readings in Distributed Computing Systems", *IEEE Computer Society Press*, 1994

[12] The Network Simulator: http://www.isi.edu/nsnam/ns/

[13] K. Petersen, M. J. Spreitzer, D. B. Terry, M. M. Theimer, and Demers, "Flexible Update Propagation for Weakly Consistent Replication", *Proceedings of the 16th ACM Symposium on Operating Systems Principles (SOSP-16)*, Saint Malo, France, October 5-8, 1997, pages 288-301.

[14] V. Duvvuri, P. Shenoy and R. Tewari, "Adaptative Leases: A Strong Consistency Mechanism for the World Wide Web", *IEEE INFOCOM 2000*, pages 834-843.