An Efficient and Fault-Tolerant Update Commitment Protocol for Weakly Connected Replicas

João Barreto[⋆] and Paulo Ferreira[⋆⋆]

INESC-ID / IST
Rua Alves Redol, 9, 1000 Lisboa, Portugal
{joao.barreto, paulo.ferreira}@inesc-id.pt

Abstract. Mobile and other loosely-coupled environments call for decentralized optimistic replication protocols that provide highly available access to shared objects, whilst ensuring an eventual convergence towards a strongly consistent state. In this paper we propose a novel epidemic weighted voting protocol for achieving such goal. Epidemic weighted voting approaches eliminate the single point of failure limitation of primary commit approaches. Our protocol introduces a significant improvement nover other epidemic weighted voting solutions by allowing multiple, causally related updates to be committed at a single distributed election round. We demonstrate that our proposed protocol is especially advantageous with the weak connectivity levels that characterize mobile and other loosely-coupled networks. We support such assumptions by presenting comparison results obtained from side-by-side execution of reference protocols in a simulated environment.

1 Introduction

Data replication is a fundamental mechanism for most distributed systems for performance, scalability and fault tolerance reasons. In particular, optimistic replication protocols [1] are of extreme importance in mobile and other loosely-coupled network environments. The nature of these environments calls for decentralized replication protocols that are able to provide highly available full access to shared objects. Such requirement is accomplished by optimistic replication strategies, which, in contrast to their pessimistic counterparts, enable updates to be issued at any one replica regardless of the availability of other replicas.

As a trade-off, the issue of consistency in optimistic replication is problematic. Since replicas are allowed to be updated at any time and circumstance, updates may conflict if issued concurrently at distinct replicas. Some optimistic replication protocols ensure that, from such a possibly inconsistent *tentative* state, replicas evolve towards an eventual consistent *stable* state. For this end, a distributed consensus algorithm is executed so as to reach an agreement on a common order in which tentative updates should be committed.

[⋆] Funded by FCT Grant SFRH/BD/13859

[⋆⋆] Funded by FCT Project UbiRep (POSI/CHS/47832/2002)

There are many scenarios where users, in order to benefit from high availability, are willing to work with temporarily tentative data, provided that a commitment agreement regarding such data will eventually be reached. Consider, for instance, a laptop user that becomes disconnected from his corporate file server after leaving his office. If necessary, he may expect to be able to modify a report that is currently replicated at his laptop, even if tentatively.

Furthermore, such worker may meet other mobile team colleagues carrying their replicas and, in an ad-hoc fashion, establish a short term work group to collaboratively work on the report. A set of causally related tentative updates will result from such activity. Hopefully, if no update is concurrently issued from outside the group, such tentative work will be eventually committed by the underlying consistency protocol. Hence, the high availability provided by an optimistic replication strategy is especially interesting in such scenarios as the previous ones. However, the usefulness of one such approach strongly depends on the ability of the underlying replication protocol to efficiently achieve a commitment decision concerning the tentatively issued data. Users are typically not inclined towards working on tentative data unless they trust the protocol to rapidly achieve a strongly consistent commitment decision regarding such data.

Aiming at such central objective, this paper proposes a novel optimistic replication protocol for efficient and highly available update commitment through the use of an epidemic weighted voting protocol based on version vectors [2]. The use of a voting approach eliminates the single point of failure of primary commit approaches [3]. Hence, the unavailability of any individual replica is not prohibitive of the progress of the update commitment process. Moreover, commitment agreement is accomplished without the need for a plurality quorum of replica servers to be simultaneously accessible: voting information flows epidemically between replicas and update commitment is based solely on local information.

The solution we propose has the main contribution of introducing a significant improvement over basic epidemic weighted voting solutions by allowing multiple update candidates to participate in an election. By using version vectors, candidates consisting of one or more causally related updates may be voted and committed by running a single distributed election round. As a result, the overall number of anti-entropy sessions required to commit updates is decreased when compared to a basic weighted voting protocol. Hence, update commitment delay is minimized and so eventual strong consistency guarantees are more rapidly delivered to applications. Namely, such reduction is substantial in scenarios where frequent causally related updates are tentatively generated by applications. The examples presented above are representative of such update patterns. In worst case scenarios, our protocol behaves similarly to basic weighted voting protocols.

The paper is organized as follows. Section 2 describes related work, Section 3 introduces the protocol, evaluated in Section 4, and Section 5 concludes.

2 Related Work

The issue of optimistic data replication for mobile and loosely coupled environments has been addressed by a number of projects [1], with the common intent of offering high data availability. Most of the proposed solutions share the goal of our work by enforcing eventual convergence towards a strongly consistent stable form that is explicitly presented to applications.

Three main approaches can be distinguished. Firstly, Golding [4] proposes that each individual server commits an update when it is certain that it has been received by every replica. A main limitation is that the unavailability of any single replica stalls the entire commitment process. On the other hand, a primary commit strategy, such as the one adopted by Bayou [3], centralizes the commitment process in a single distinguished primary replica that establishes a total commit order over the updates it receives. Primary commit is able to rapidly commit updates, since if suffices for an update to be received by the primary replica to become committed, provided that no conflict is found. However, should the primary replica become unavailable, the commitment progress of updates generated by replicas other than the primary is inevitably halted.

Finally, a third approach uses voting so as to allow a plurality quorum to commit an update. In particular, Deno [5] relies on an epidemic voting protocol to support object replication in a transactional framework for loosely-connected environments. Deno requires one entire election round to be completed in order to commit each single update, if only non-commutable updates are considered. This is acceptable when applications are interested in knowing the commitment outcome of each tentatively issued update before issuing the next one. However, in the usage scenarios addressed by this paper, users and applications will often be interested in issuing multiple tentative updates before acknowledging their commitment. In such situations, the commitment delay imposed by Deno's voting protocol becomes unacceptably higher than that of primary commit.

3 Consistency Protocol

The following sections consider a model where a set of logical objects is replicated at multiple server hosts. An object replica at a given server provides local applications with access to a version of the object contents, as stored by the replica. Such accesses may read or modify the object contents. In the case of the latter, an update is issued by the server and applied to the replica.

Updates issued at a given replica are propagated to other servers in an epidemic fashion in order to eventually achieve object consistency. The local execution of an update is assumed to be recoverable, atomic and deterministic. The former means that a replica will not reach an inconsistent value if it fails before the update execution completes. It follows from the other two properties that the execution of the same ordered sequence of updates at two distinct replicas in the same initial consistent state will yield an identical final state. For simplicity and without loss of generality, we consider that each logical object is replicated at every server in the system. For the sake of generality, the set of replicas may be dynamic, and thus change with the creation or removal of new servers.

Hereafter, we assume an asynchronous system in which servers can only fail silently. Network partitions may also occur, thus restricting connectivity between servers that happen to be located in distinct partitions.

3.1 Overview

Due to the optimistic nature of the consistency protocol, an update issued at a local replica is not immediately committed at every remaining replica. Instead, such update is considered to be in a tentative form since conflicting updates may still be issued at other replicas. The consistency protocol is responsible for committing such tentative updates into a total order that will be eventually reflected at every replica.

Our protocol achieves this goal through a weighted voting approach [5]: concurrent tentative updates are regarded as rival candidates in an election. The servers replicating a given logical object act as voters whose votes determine the outcome of each election between candidate updates to the object. A candidate update wins an election by collecting a plurality of votes, in which case it is committed and its rival candidates are discarded.

Elections consider a fixed per-object currency scheme, in which each voter is associated with a given amount of currency that determines its weight during voting rounds. The global currency of a logical object, distributed among its replica servers, equals a fixed amount of 1. Currencies can be exchanged between servers and the currency held by failed servers can be recovered by running a *currency reevaluation* election, as discussed in [6].

Version Vector Candidates In some cases, applications will be interested in generating more than one tentative update prior to its commitment decision. These may include disconnected mobile applications and ad-hoc groups of mobile applications working cooperatively in the absence of a plurality quorum. Since the commitment decision may not be taken in the short-term, these applications may wish to issue a sequence of multiple, causally ordered tentative updates.

In order to efficiently accommodate for such update scenarios, the novel solution proposed in this paper employs version vectors to identify candidate updates in a weighted voting protocol. The flexibility brought by version vectors allows a sequence of one or more updates to run for the current election as a whole. In this case, the candidate is represented by the version vector corresponding to the tentative version obtained if the entire update sequence was applied to the replica. As the next sections explain, the voting protocol relies on the causality expressiveness of version vectors to deciding if the update sequence or a prefix of it are to become committed. Consequently, candidates consisting of one or more causally related updates may be committed on a single distributed election round. In weakly connected network environments, where such update patterns are expectably dominant, a substantial reduction of the update commitment delay is therefore achievable.

Each replica r maintains the following state:

- $stable_r$, which consists of a version vector that identifies the most recent stable version that is currently known by replica r, obtained after the ordered application of all committed updates;

- $votes_r[1..N]$, which stores, for each server $k = 1, 2, .., N$, the version vector corresponding to the candidate voted for by k, as known by r; or $\perp$, if the vote of such server has not yet been known to r;
- $cur_r[1..N]$, which stores, for each server $k = 1, 2, .., N$ whose vote replica r has knowledge of, the currency associated with such vote;

Each server is able to offer two possibly distinct views over the value of a replica r to its applications and users: the stable and tentative views. The first view reflects a strongly consistent value of the replicated object that is identified by $stable_r$. On the other hand, the tentative view exposes a weakly consistent value that corresponds to the candidate version that is currently voted by the local server, $votes_r[r]$.

Issuing a tentative update on a replica r causes a new candidate to run for the current election according to the following rules:

1. If $votes_r[r] = \perp$, then $votes_r[r] \leftarrow adv_r(stable_r)$[1] and $cur_r[r] = currency_r$;
2. Otherwise, $votes_r[r] \leftarrow adv_r(votes_r[r])$;

As the next sections describe in greater detail, voting information flows in an epidemic fashion among servers and the decision to commit an update is based only on local replica information. These are important properties for operation under mobile and loosely-coupled environments. In particular, Section 3.2 addresses the storage of tentative update and their corresponding commitment upon a replica value. Section 3.3 then describes the epidemic flow of consistency information and Section 3.4 finally defines how candidates are elected.

3.2 Update Commitment

The protocol proposed hereafter is orthogonal to the issues of actual transference and storage of tentative updates. In particular, the protocol does not impose the decision of whether to transfer and store, at each individual replica, the tentative updates belonging to every candidate in the current election or, alternatively, only those concerning the replica's own candidate.

This means that, at the time a server determines that a given candidate has won the election and, thus, its updates should be committed, such updates may not be immediately available. Instead, they will be eventually collected through succeeding anti-entropy sessions with other servers. Consequently, there may occur a discrepancy between the most recent stable version identified by the consistency protocol and the actual stable value that is locally accessible. Such discrepancy is enabled by an additional element at the state of each replica r:

- c_r, which consists of an integer value representing the number of updates in the stable path that have already been committed by replica r;

The value of c_r may be lower than the number of updates that have actually been determined by the consistency protocol as belonging to the stable path. In such case, the replica's stable value does not yet reflect the most recent stable

[1] adv_r advances the counter corresponding to r in the supplied version vector by one.

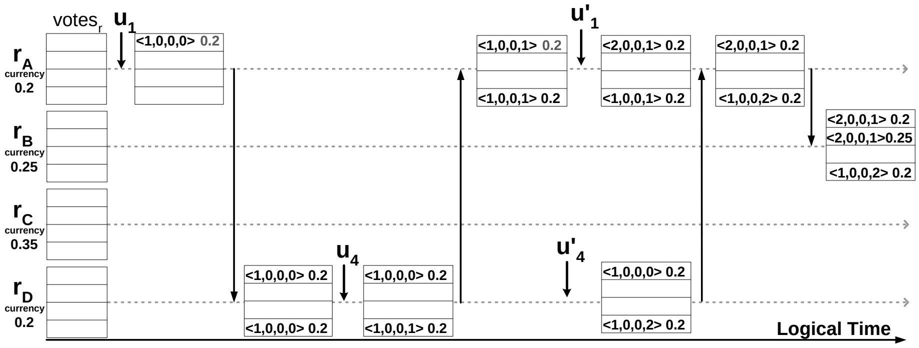

Fig. 1. Example of update generation and propagation: four replicas with unevenly distributed currencies start from a common initial stable version $stable_r = \langle 0, 0, 0, 0 \rangle$.

version r is aware of. As a consequence, the protocol is flexible enough to support servers with differing memory limitations.

On one hand, servers with rich memory resources may store every update associated with each candidate, hence being able to immediately gain access to the most recent known stable value as each new stable version is determined by the protocol. On the other hand, memory-constrained devices may opt to restrict themselves to storing only the updates of their own candidate and, thus, allow for occasional delays in the availability of the most recent stable value when rival candidates win an election. In either case, however, the efficiency of the protocol in taking commitment decisions is not affected. Both strategies may transparently co-exist in a system of replicas of the same logical object.

Moreover, it is assumed that a log of committed updates is maintained, including the following information:

- $gen_r[1..c_r]$, which stores, for each update committed so far in replica r, the server that generated it.

From the consistency protocol's viewpoint, the procedure for committing a sequence of updates $u_1, .., u_n$, generated by servers $i_1, ..i_n$, respectively, is therefore comprised by the following steps:

1. For each update, u_k, $gen_r[c_r + k] \leftarrow i_k$;
2. $c_r \leftarrow c_r + n$;

3.3 Anti-entropy

Voting information is propagated through the system by anti-entropy sessions established between pairs of accessible replicas. An anti-entropy session is an unidirectional pull-based interaction in which a requesting replica, A, updates its local election knowledge with information obtained from another replica, B. In case B has more up-to-date election information, it transfers such information to A. Furthermore, if A has not yet voted for a candidate that is concurrent to the one voted for by B, A accepts the latter, thus contributing to its election.

Each anti-entropy session is carried out according to the following procedure, which should be executed atomically:

1. If $stable_A < stable_B$ then
   (a) $stable_A \leftarrow stable_B$;
   (b) $\forall k$ s.t. $votes_A[k] \| stable_A$ or $votes_A[k] \leq stable_A$, then $votes_A[k] \leftarrow \perp$;
2. If ($votes_A[A] = \perp$ and $stable_A < votes_B[B]$) or $votes_A[A] < votes_B[B]$ then $votes_A[A] \leftarrow votes_B[B]$ and $cur_A[A] \leftarrow currency_A$;
3. $\forall k \neq A$ s.t. ($votes_A[k] = \perp$ and $stable_A < votes_B[k]$) or $votes_A[k] < votes_B[k]$, then $votes_A[k] \leftarrow votes_B[k]$ and $cur_A[k] \leftarrow cur_B[k]$.
4. If $c_A < c_B$ then commit update sequence issued by $gen_B[c_A+1], .., gen_B[c_B]$.

The first step ensures that, in case r_B knows about a more recent stable version, r_A will adopt it. This means that r_A will regard the elections that originated such new stable version as completed and so begin a new election from that point. Such new election is prepared by keeping only the voting information that will still be meaningful for the outcome of the election. Namely, these are the votes on candidates that causally succeed the stable version.

As a second step, r_A is persuaded to vote for the same candidate as the one voted by r_B, provided that r_A has not yet voted for a concurrent candidate. Subsequently, r_A updates its current knowledge of the current election with relevant voting information that may be held by r_B. Namely, r_A stores each vote that it is not yet aware of or whose candidate is more complete than the one it currently has knowledge of.

Finally, the set of committed updates held by B that are not yet locally available at replica A are collected and committed by the latter. An example of update generation and propagation through anti-entropy is illustrated in Fig. 1.

3.4 Election decision

The candidates being voted in an election represent update paths that traverse through one of more versions beyond the initial point defined by the stable version, *stable*. These possibly divergent candidate update paths may share common prefix sub-paths. The following definition expresses such notion.

Definition 1: *Maximum common version.* Given two version vectors, v_1 and v_2, their maximum common version is given by a version vector, $mcv(v_1, v_2)$, s.t. $\forall k, mcv(v_1, v_2)[k] = min(v_1[k], v_2[k])$. For simplicity, we represent $mcv(v_1, v_2, .., v_m)$ as the result of $mcv(mcv(mcv(v_1, v_2)), ...), v_m)$.

Theorem 1: Let $v_1, .., v_m \in votes_r$, be one or more candidate versions known by replica r, each connoting a tentative update path starting from the stable version, $stable_r$. Their maximum common version, $mcv(v_1, .., v_m)$, constitutes the farthest version of an update sub-path that is mutually traversed by the update paths of $v_1, .., v_m$. Complementarily, the total currency voted for such common sub-path is obtained by $voted_r(mcv(v_1, .., v_m)) = cur_r[1] + ... + cur_r[N]$.

The voting protocol is responsible for progressively determining common sub-paths of candidate versions that manage to obtain a plurality of votes. This decision is based on the definition of maximum common version among the set of candidate versions voted at a given replica and on the value of uncommitted currency, $uncommitted_r = \sum cur[k] : votes_r[k] \neq \perp$, according to the following:

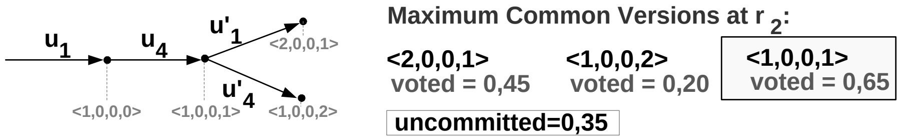

Fig. 2. Election decision for replica r_2 at the final state in Figure 1. Candidate $\langle 1,0,0,1\rangle$ has collected a plurality of votes and, thus, u_1 and u_4 will be committed in that order.

Definition 2: Let w be a version vector s.t. $w = mcv(w_1, .., w_m)$ where $w_1, .., w_m \in votes_r$ and $1 \leq m \leq N$. w wins an election when:

1. $voted_r(w) > 0.5$, or
2. $\forall l$ s.t. $l = mvc(l_1, .., l_k), l_1, .., l_k \in votes_r$, $1 \leq k \leq N$ and $l \parallel w$,
   (a) $voted_r(w) > voted_r(l) + uncommited_r$, or
   (b) $voted_r(w) = voted_r(l) + uncommited_r$ and $w <_{lex} l$.

The above rules state the conditions that guarantee that a candidate has collected sufficient votes to win an election. The votes may constitute a majority, when the amount of currency voted on the winning candidate surpasses 0.5; or a simple plurality, when the voted currency is greater than the maximum potentially obtainable currency of any other rival candidate. Ties are decided by choosing the candidate whose version vector is lexically lower. If one represents each version vector as a number whose digits are the elements of the vector, such representation can be numerically compared, thus inducing a lexical order, $<_{lex}$, in the version vector space.

Determining if a candidate has won an election depends exclusively on information that is locally available at each replica. This means that, once having collected enough voting information, a given replica is able to decide, by its own, to commit a candidate version that locally fulfills the election winning conditions. Hence, update commitment is accomplished in a purely decentralized manner. An example is depicted in Fig. 2.

After finding a new winner version vector, w, a replica r atomically takes the following steps to accept the election decision and prepare for the next election:

1. $stable_r \leftarrow w$;
2. $\forall v_k \in votes_r$ s.t. $v_k \| w$ or $v_k \leq w$, $votes_r[k] \leftarrow \perp$;
3. If the sequence of updates that comprise the update path defined between versions $stable_r$ and w is locally available, then commit it;

After accepting the election result by setting the winning version as the new stable version, the second step resets all the defeated candidates to $\perp$. Depending on the local availability of the updates that belong to the winning candidate, they may be committed into the replica's stable value; otherwise, further anti-entropy sessions will ensure that such updates are eventually collected and committed. A new election can then take place.

Theorem 2 (Correctness): After all elections have been completed at every replica and all updates belonging to the resulting stable path have been committed at every replica: $\forall r, t$, replica r has committed the same ordered sequence of updates as t.

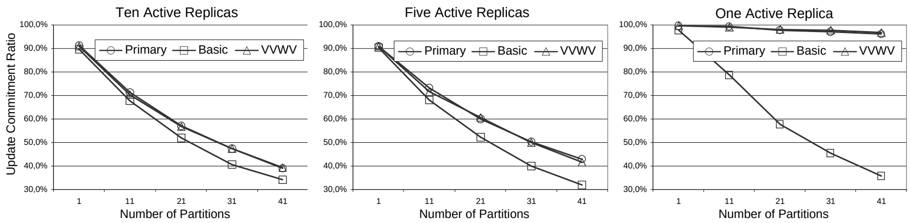

Fig. 3. Update commitment ratios versus number of partitions, for different numbers of active replicas.

4 Evaluation

C# implementations of the primary commit (*Primary*), basic weighted voting (*Basic WV*) and version vector weighted voting (*VVWV*) protocols were run side-by-side in a simulated environment. The simulator includes a collection of replicas of a common logical object, randomly distributed by a set of network partitions. Time is divided into logical time slices; at each time slice, each replica (1) with a given *mobility probability*, migrates to a different, randomly chosen, network partition; (2) pulls anti-entropy information from a partner, randomly selected from the set replicas present in its current partition; and, (3) generates, with a given *update probability*, one tentative update. Each replica may be active or inactive; in the case of the latter, its update probability is null. An inactive replica exchanges, with a given *activation probability*, its status with an active replica after pulling anti-entropy information from it. The differentiation between active and inactive replicas allows for non-uniform update models to be simulated, namely the hot-spot model [7], which assumes, based on empirical evidence, that updates typically occur in a small set of replicas.

The protocols were evaluated against an increasing number of partitions. Since update contention is prone to arise in a partitioned system, the update commitment delay is not a sufficiently meaningful measure for our purposes, as it does not take into account the discarded updates. Instead, a better evaluation is provided by the update commitment ratio of each protocol, i.e. the percentage of issued updates that is committed at all replicas.

The measurements were obtained with the fixed settings of 10 replicas with mobility and activation probabilities of 20% and 40%, respectively, running for 2000 time slices on each experiment; we observed that the variation of such values does not have a relevant impact on obtained results. Three update models were tested: with ten, five and just a single active replicas; a global update probability of 5%, evenly divided by the active replicas, was considered.

The commitment ratio is directly affected by the efficiency of each evaluated update commitment protocol, since if updates remain in their tentative state for longer periods, the probability of conflicts is higher; hence, lower commitment ratios reflect longer delays imposed by the update commitment process. So, as expected, update commitment ratios decrease as the connectivity among replicas is weakened by an increasing number of partitions, as shown in Figure 3.

However, Primary and VVWV are able to ensure higher ratios than Basic WV as partitioning grows. Situations of multiple causally related tentative updates occur more frequently as updates remain tentative for longer periods. Hence, such results are explained by the efficiency of the former protocols in the commitment of multiple causally related updates, in contrast to Basic WV. Such situations are also increased as the global update probability is distributed by a smaller number of active replicas. Accordingly, the advantage of Primary and VVWV over Basic WV is accentuated as the number of active replicas decreases. It should be noted that higher update probabilities yielded equivalent, yet magnified, conclusions. On the other hand, Primary and VVWV have similar ratios; however, VVWV has the crucial advantage of not depending on a single point of failure.

Finally, similar experiments compared the two update storage alternatives of VVWV. A maximum improvement of 0.8% was attained by storing the updates of all candidates, which suggests that the more resource-efficient alternative of storing only the updates of a replica's own candidate is acceptable.

5 Conclusions

We propose a novel epidemic weighted voting protocol, VVWV, for achieving the goal of optimistic update commitment that allows multiple causally ordered update candidates to be committed at a single election round. Simulation results show that, under weak connectivity conditions, VVWV is advantageous relatively to a basic weighted voting protocol, while attaining similar update commitment ratios to the less fault-tolerant primary commit protocol.

Additional work [8], not addressed in this paper, shows how dynamic version vector maintenance can be effectively incorporated into the proposed protocol and proves Theorems 1 and 2.

References

1. Saito, Y., Shapiro, M.: Optimistic replication. ACM Comput. Surv. **37** (2005) 42–81
2. Parker, D.S., et al: Detection of mutual inconsistency in distributed systems. Distributed systems, Vol. II: distributed data base systems (1986) 306–312
3. Petersen, K., et al: Flexible update propagation for weakly consistent replication. In: Proceedings of the 16th ACM Symp. on Operating Systems Principles. (1997)
4. Golding, R., Long, D.: Modeling replica divergence in a weak-consistency protocol for global scale dirstibuted data bases. Technical Report UCSC-CRL-93-09 (1993)
5. Keleher, P.: Decentralized replicated-object protocols. In: Proc. of the 18th Annual ACM Symp. on Principles of Distributed Computing (PODC'99). (1999)
6. Cetintemel, U., Keleher, P.: Light-weight currency management mechanisms in mobile and weakly-connected environments. Dist. Par. Databases **11** (2002) 53–71
7. D. Ratner, P.R., Popek, G.: Roam: A scalable replication system for mobile computing. In: Mobility in Databases and Distributed Systems. (1999)
8. Barreto, J., Ferreira, P.: Optimistic consistency with dynamic version vector weighted voting. Technical Report RT/008/2004, Inesc-ID (2004)