Adaptive Voting for Balancing Data Integrity with Availability

Johannes Osrael, Lorenz Froihofer, Matthias Gladt, Karl M. Goeschka

Vienna University of Technology
Institute of Information Systems
Argentinierstrasse 8/184-1, 1040 Wien, Austria
[osrael|froihofer|gladt|goeschka]@tuwien.ac.at

Abstract. Data replication is a primary means to achieve fault tolerance in distributed systems. Data integrity is one of the correctness criteria of data-centric distributed systems. If data integrity needs to be strictly maintained even in the presence of network partitions, the system becomes (partially) unavailable since no potentially conflicting updates are allowed on replicas in different partitions. Availability can be enhanced if data integrity can be temporarily relaxed during degraded situations. Thus, data integrity can be balanced with availability.
In this paper, we contribute with a new replication protocol based on traditional quorum consensus (voting) that allows the configuration of this trade-off. The key idea of our Adaptive Voting protocol is to allow non-critical operations (that cannot violate critical constraints) even if no quorum exists. Since this might impose replica conflicts and data integrity violations, different reconciliation policies are needed to re-establish correctness at repair time. An availability analysis and an experimental evaluation show that Adaptive Voting provides better availability than traditional voting if (i) some data integrity constraints of the system are relaxable and (ii) reconciliation time is shorter than degradation time.

1 Introduction

Replication is one of the primary mechanisms to enhance availability of distributed systems. One correctness criterion for data-centric applications are data integrity constraints, such as value constraints, relationship constraints (cardinality, XOR), uniqueness constraints and other predicates. A system is *constraint consistent* if all data integrity constraints are satisfied.

If strict constraint consistency has to be ensured all the time - even in the presence of failures - the system becomes (at least partially[1]) unavailable in degraded scenarios (e.g., node or link failures) since neither potentially conflicting updates on replicas in different partitions nor updates that possibly violate data integrity constraints are allowed. On the other hand, some applications (e.g., [3, 4]) exist where consistency can be temporarily relaxed in order to achieve higher availability.

The DeDiSys [5] middleware extends state-of-the-art object-based middleware such es CORBA, EJB, or .NET with explicit management of data integrity constraints and

[1] Even if a majority partition or more generally - a quorum - exists [1, 2], significant parts of the system become unavailable.

provides novel adaptive replication protocols that enable the balancing between availability and consistency.

In this paper[2], we focus on the replication part of the DeDiSys middleware and contribute with a new replication protocol called *Adaptive Voting* (AV) that allows to configure the trade-off between availability and data integrity. AV is based on the traditional voting scheme [2] but allows non-critical operations even if no quorum exists. AV offers different policies to re-establish correctness when the failures are repaired.

Paper Overview Our system model is presented in Sect. 2. Section 3 introduces the key idea of the new Adaptive Voting protocol. Section 4 describes the different modes of AV in detail. Section 5 presents the results of an experimental evaluation of the protocol. Our work is compared to related work in Sect. 6 before we conclude in Sect. 7.

2 System Model

We focus on tightly-coupled, data-centric, object-oriented distributed systems with up to about 30 server nodes and an arbitrary number of client nodes. Server nodes host objects which are replicated to other server nodes in order to achieve fault tolerance. We assume full replication, i.e., objects are replicated to all nodes. We consider both node and link failures (partitioning): The crash failure [6] model is assumed for nodes and links may fail by losing but not duplicating or corrupting messages.

We assume a partially synchronous system, where clocks are not synchronized, but message time is bound. A group membership service is assumed in our system, which provides a single view of the nodes within a partition, i.e., it is used to detect node and link failures. Furthermore, we assume the presence of a group communication service which provides multicast to groups with configurable delivery and ordering guarantees.

We assume the correctness of the system is expressed in the form of application-specific data integrity constraints, which are defined upon objects that encapsulate application data (e.g., Entity Beans in Enterprise Java Beans terminology). These objects do not contain business logic and typically correspond to a row in a table of a relational database.

Not all constraints of an application are of equal importance. Some have to be satisfied at any point in time while others might be relaxed temporarily when failures occur. To allow such flexibility, we provide two different constraint classes [7] with respect to the trading of constraint consistency for availability: *Non-tradeable constraints* must never be violated. Thus they cannot be traded for higher availability during degradation. *Tradeable constraints* can be temporarily relaxed during degraded situations. Read operations cannot affect data integrity constraints; thus they are non-critical with respect to data integrity. For write operations we distinguish *critical* and *non-critical* write operations. The former affect at least one non-tradeable while the latter affect only tradeable constraints.

[2] This work has been partially funded by the European Community under the FP6 IST project DeDiSys (Dependable Distributed Systems, contract number 004152, http://www.dedisys.org).

3 Adaptive Voting

3.1 Weighted Voting

In Weighted Voting [2], a generalization of Majority Voting [1], each replica is assigned some number of votes. Whenever a read or write operation shall be performed, at least RQ (read quorum) or WQ (write quorum) votes must be acquired. Let the total number of votes be V. The following conditions must be satisfied:

$$RQ + WQ > V \qquad (1) \qquad\qquad WQ > \frac{V}{2} \qquad (2)$$

$WQ, RQ, V \in \mathbb{N}$ and $WQ, RQ \leq V$ are assumed[3]. Condition (1) prevents read-write conflicts while condition (2) prevents write-write conflicts.

For the sake of simplicity we further assume in this paper that all replicas have equal votes (i.e., 1) and each node in the system hosts one replica. Thus, the total number of votes V becomes the total number of nodes in the system, denoted as N. We denote this simplification of weighted voting as *Traditional Voting*.

Quorum consensus techniques allow to balance the cost of read against write operations by adjusting the sizes of the read and write quorum appropriately. Furthermore, in static quorum schemes (as weighted voting), where the quorums are not reconfigured in response to failures, no intervention is necessary when network failures are repaired or nodes recover; i.e., failures are masked.

3.2 Key Concept of Adaptive Voting

Traditional voting blocks operations if the quorums cannot be built. However, as discussed in Sect. 1, some systems do not require strict data integrity all times, i.e., constraint consistency can be temporarily relaxed during degraded situations.

Thus, our key idea is to enhance availability of traditional voting by allowing non-critical operations even if no quorums exist, i.e., operations are allowed that may violate tradeable constraints but do not affect non-tradeable constraints. Furthermore, the new protocol called *Adaptive Voting* (AV) allows to re-adjust the quorums in degraded situations in order to support the tuning[4] of read against write operations. Since update conflicts and data integrity violations might be introduced, different policies are required to re-establish replica and constraint consistency after nodes rejoin. The replica consistency requirement for quorum consensus protocols is that a write quorum of replicas is consistent.

AV distinguishes three modes of operation: normal mode, degraded mode, and reconciliation mode. The latter can be further divided into two sub-modes: replica consistency reconciliation and constraint consistency reconciliation. The current mode of the replication protocol depends on the system state.
AV is in the *normal mode* when all nodes are reachable and all constraints are satisfied, i.e., no partitions are present and all repair activities (reconciliation) are finished.

[3] In this paper, we denote with $\mathbb{N}$ all positive natural numbers, i.e., zero is not included.

[4] The choice of the quorums depends on the read/write ratio and is not influenced by the data integrity constraints.

The replication protocol switches into the *degraded mode* when not all nodes are reachable. Since node and link failures cannot be distinguished [8], node failures are treated as network partitions until repair time.

AV enters *reconciliation mode* when two or more partitions rejoin. The objective of reconciliation is to re-establish replica and constraint consistency of the system. System-wide constraint consistency can only be guaranteed if all nodes are reachable (i.e., after the last rejoin of partitions). Thus, if partitions rejoin but the merged partition does not contain all nodes, only replica consistency is re-established. When the last partition rejoins[5], both replica and constraint consistency can be re-established.

4 Protocol Description

4.1 Normal Mode

In normal mode, AV behaves as the traditional voting protocol with the enhancement that constraints are checked in case of write operations.

4.2 Degraded Mode

In traditional voting, read operations are allowed if a read quorum can be acquired and write operations if a write quorum can be acquired.

AV enhances availability by allowing non-critical operations even if no quorum exists. Critical operations are treated as in the normal mode to ensure that non-tradeable constraints are never violated.

Partition Size The behavior of AV in degraded mode depends on the size of the partition. The number of nodes within a partition is denoted as P. Thus the number of nodes outside the partition is $N - P$. We denote the quorum sizes of the healthy system (i.e., all nodes are reachable) as WQ_H and RQ_H. For the following considerations we assume that the quorums are not larger than necessary, i.e., $WQ_H + RQ_H = N + 1$, which further implies $RQ_H \leq WQ_H$.

Write and read quorum exist: If a write quorum exists in a partition, a read quorum exists as well. Outside the partition, no read or write quorum can exist.

$$N > P \geq WQ_H \geq RQ_H \qquad \Leftrightarrow \qquad N - P < RQ_H \leq WQ_H \tag{3}$$

Both critical and non-critical operations are allowed. For critical operations, the behavior is as in the normal mode. We allow non-critical updates in other partitions even if no write quorum exists. Thus, the quorum conditions are no longer satisfied system-wide and write-write or read-write conflicts might arise. However, we avoid partition-internal

[5] In our target application scenarios, nodes either eventually rejoin or are explicitly excluded from the system after a while, e.g., by a system administrator.

conflicts by using a quorum scheme within the partition. The (partition-internal) quorums can be adjusted according to the size of the partition. We denote the reduced quorums in the partition as WQ_P and RQ_P.

Since a read quorum RQ_H as defined in the healthy system exists, up-to-date copies of objects affected by non-tradeable constraints can be retrieved. For objects affected by tradeable constraints, the read quorum might have been reduced in the partition. Thus, performance of the read operation can be improved by reading from RQ_P. However, since updates on objects affected by tradeable constraints are allowed in all partitions, the read operation might return an object that is *possibly stale*.

Write quorum does not exist but read quorum exists: If a read quorum but no write quorum exists in the partition, outside the partition no write quorum can exist but a read quorum may exist:

$$WQ_H > P \geq RQ_H \qquad \Leftrightarrow \qquad WQ_H > N - P \geq RQ_H \tag{4}$$

Only non-critical operations are allowed in this situation. As mentioned before, the quorum sizes can be reduced for non-critical operations since valid quorums are guaranteed only within the partition anyway.
Thus, the following steps are performed:

1. Check if one of the constraints affected by the operation is non-tradeable. If yes, the update is not allowed. Otherwise proceed.
2. Find write quorum WQ_P and apply operation.
3. Mark constraints for re-evaluation: All objects affected by tradeable constraints are possibly stale in degraded mode since updates are allowed in different partitions. Thus, the constraint check needs not to be performed since it has no significance in general. However, the constraint is marked for re-evaluation at reconciliation time.

An object is saved in a version history before it is changed in degraded mode. This allows detection of update conflicts and stepwise rollback during reconciliation in case of constraint consistency violations.
Read operations are treated as in case (3).

Write and read quorum do not exist: If no read and write quorum exist in the partition, both a read and write quorum may exist outside the partition:

$$RQ_H > P \geq 1 \qquad \Leftrightarrow \qquad N - P \geq WQ_H \geq RQ_H \tag{5}$$

Write operations are treated as in case (4). Read operations can only be performed on a reduced read quorum RQ_P. Thus, all objects returned by a read operation are possibly stale. However, by applying the quorum conditions in the partition, it is guaranteed that subsequent read operations within a partition will return the same version.

Quorum Adjustment AV allows updates in different partitions during degraded situations. However, within a partition, read-write and write-write conflicts shall be prevented and the tuning of read against write operations shall be supported. Thus, a quorum scheme adapted to the size of the partition is applied:

$$WQ_P + RQ_P > P \qquad (6) \qquad WQ_P > \frac{P}{2} \qquad (7)$$

$$WQ_P, RQ_P, P \in \mathbb{N} \qquad (8) \qquad WQ_P, RQ_P \leq P \qquad (9)$$

Different quorum adjustment policies can be distinguished:

Adjustment Policy 1: Maintaining read and write quorum The most obvious strategy is to maintain the quorum sizes as in the healthy system as long as possible. If the partition size P falls below WQ_H (RQ_H), the write (read) quorum is set to P:

$$WQ_P = \min(WQ_H, P) = \begin{cases} WQ_H & : \quad P \geq WQ_H \\ P & : \quad P < WQ_H \end{cases} \qquad (10)$$

$$RQ_P = \min(RQ_H, P) = \begin{cases} RQ_H & : \quad P \geq RQ_H \\ P & : \quad P < RQ_H \end{cases} \qquad (11)$$

Adjustment Policy 2: Proportional adjustment Adjustment policy 1 maintains the configuration of the healthy system as long as possible but with the cost that the quorums become larger than necessary. In order to maintain the read/write tuning of the healthy system, the read and write quorum in the partition can be adjusted proportional to the size of the partition. Since $RQ, WQ \in \mathbb{N}$, exact proportional adjustment is not always possible. However, to achieve "optimal" proportional adjustment,

$$\min_{RQ_P, WQ_P} \quad \left| \frac{P}{N} - \frac{WQ_P}{WQ_H} \right| + \left| \frac{P}{N} - \frac{RQ_P}{RQ_H} \right| \qquad (12)$$

needs to be solved, considering the above mentioned side conditions (6), (7), (8), (9), and $WQ_P + RQ_P = P + 1$ to minimize the quorum sizes. This discrete linear optimization problem can be solved e.g., by using the branch and bound algorithm [9].

Adjustment Policy 3: Arbitrary adjustment In principle, all adjustments are allowed, as long as the above mentioned conditions (6), (7), (8), and (9) are met.

Figure 1 gives examples for the presented quorum adjustment strategies. The horizontal axis denotes the size P of the partition, decreasing from the left side ($P = N$) to the right side (P=1). The vertical axis shows the sizes of the read and write quorums.

4.3 Reconciliation Mode

The overall goal of reconciliation is to re-establish constraint consistency. However, full constraint consistency can only be re-established if all nodes are available. Thus, if this is not the case, AV only re-establishes replica consistency in the merged partition.

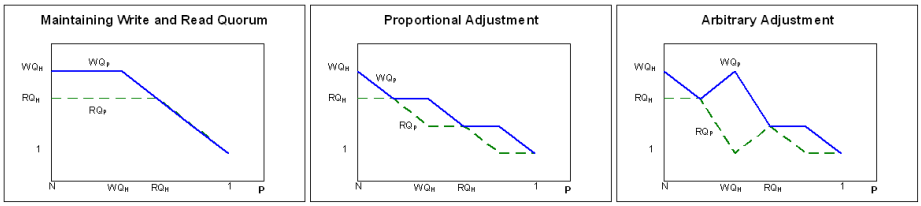

Fig. 1. Quorum adjustment policies

Non-critical operations are not allowed in reconciliation mode, therefore availability is reduced in this mode. Thus, reconciliation should be as fast as possible. In order to avoid combinatorial explosion, we use simple (application-defined) heuristics in the reconciliation phase, e.g., to select a particular version in case of a write-write conflict. Reconciliation is performed in the following steps:

1. Re-adjustment of quorum sizes.
2. Re-establishment of replica consistency.
3. Re-establishment of constraint consistency if all nodes are available.

Re-adjustment of quorum sizes The quorum sizes of the merged partition needs to be adjusted so that the appropriate quorum conditions are obeyed.

Re-establishment of replica consistency AV allows non-critical updates in all partitions, even if no write quorum exists. Thus, write-write conflicts might arise. These conflicts can be detected by comparing the version lists of the partitions. If updates have occurred in only one partition, the version list of this partition is applied at a write quorum of the merged partition. In case of a conflict between the updates in the different partitions (we denote this as *replica conflict*), one of the replicas is chosen according to some pre-defined criterion (e.g., the larger partition wins). The version list of the losing partition is discarded. The version list of the winning partition is adopted by a write quorum of the merged partition.

Re-establishment of constraint consistency System-wide constraint consistency can only be re-established if all nodes are available. If this is the case, all constraints that are marked for re-evaluation are checked again. If a constraint is violated, the following policies are defined to re-establish data integrity:

1. Constraint conflict policy 1: Stepwise rollback: Objects affected by the constraint are stepwise reverted to previous versions till the constraint is satisfied.
2. Constraint conflict policy 2: Compensation actions: In order to avoid rollbacks, application-specific compensation actions can be defined. For instance, a simple compensation action is to choose a default version in case of a conflict.

The chosen version is applied at a write quorum of the merged partition. All tentative versions are discarded, i.e., the version lists are cleaned.

5 Experimental Evaluation

Based on our availability analysis [10] we have concluded that AV yields better availability over time if reconciliation time is short in comparison to degradation time[6]. Thus, we have implemented AV in the Java-based Neko framework [11] and compare reconciliation time vs. degradation time for two different constraint conflict policies. In case of a replica conflict, the version of the partition where more updates have occurred is chosen.

A simple inter-object constraint is assumed: The state of an object of class A/B is represented by an integer value a/b. For each pair of objects of class A and B, the constraint $a + b < constant$ must hold in the healthy system. The constraint is tradeable, i.e., it can be temporarily relaxed during degradation. An update operation can increment a/b by 1. The system consists of 20 nodes. The initial write quorum is 15 and the initial read quorum is 6. The system degrades into two partitions, both containing 10 nodes. The write/read quorums are adjusted to 8/3 in both partitions in order to balance read against write operations. On average, 150 invocations are started per second in each partition. a is updated in one partition and b in the other one. The total degradation time has been varied from 3 to 60 seconds. The measurements have been conducted in the Neko v0.9 simulation mode on a single machine (Pentium M, 795 MHz, 1 GB RAM, Windows XP SP2).

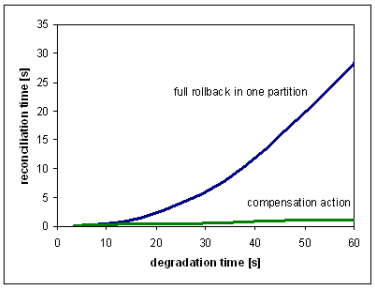

Fig. 2. Reconciliation time vs. degradation time

Figure 2 shows the lower bound of reconciliation time if a compensation action is applied and the upper bound if a roll-back strategy is applied in our example configuration. The fastest compensation action is to set default values in case of a constraint violation. If constraint consistency within the partitions is assured, the worst case for reconciliation is that a partition needs to be rolled-back completely.

The evaluation gives an indication that reconciliation time is shorter than degradation time in a configuration typical for our target applications. However, reconciliation time is highly application-specific (depending on the constraints, failure pattern, load during degradation, reconciliation policies, etc.), thus future work includes evaluation of the protocol in a real-world application scenario - the Experimental Physics and Industrial Control System (EPICS) [4].

[6] Degradation time is the period where node and/or link failures are present.

6 Related Work

Various dynamic quorum schemes (e.g., [12, 13]) have been proposed which adapt to changes in the system due to failures. However, in contrast to our approach they are (i) pessimistic, i.e., they preserve replica consistency despite failures, and (ii) do not consider data integrity as correctness criterion.

Trading replica consistency for increased availability has been addressed in distributed object systems such as [14, 15]. However, these systems either guarantee strong replica consistency or no replica consistency at all. TACT (Tunable Availability and Consistency Trade-offs) [16] fills in the space between by providing a continuous consistency model based on logical consistency units (*conits*). The consistency level of each conit is defined using three application-independent metrics – numerical error, order error, and staleness. TACT provides a fine-grained trade-off between replica consistency and availability but does not focus on constraint consistency.

While our approach treats disconnected operation as a failure scenario, disconnections are inherent in mobile environments. Thus, different solutions for reconciliation of divergent replicas have been proposed for mobile environments: In Bayou [17], application developers need to define application-specific conflict detection and reconciliation policies. Replica consistency is re-established by an anti-entropy protocol with eventual consistency guarantees. Our approach offers the same flexibility as Bayou but offers pre-defined reconciliation policies in addition. Gray et al. [18] introduced the concept of tentative transactions: Transactions are tentatively committed on replicated data on mobile (disconnected) nodes and later applied at a master copy when the nodes rejoin. If the commit on the master copy fails, the originating node is informed why it failed. Application-specific semantics are used for conflict resolution in the mobile transaction management system presented in [19].

Besides the already mentioned differences to our approach, *all* of the above replication and reconciliation approaches and other optimistic replication systems [20] have one commonality: In contrast to our approach, they either do not address constraint consistency explicitly or presume strong data integrity.

7 Conclusion

We presented Adaptive Voting (AV), a new replication protocol based on traditional voting, that allows to balance data integrity against availability in degraded situations when node and link failures occur. The key idea of AV is to allow non-critical operations (that only affect non-critical data integrity constraints) even if the quorum conditions cannot be met. AV does not only provide adaptiveness by trading availability against constraint consistency, it also allows to re-adjust the quorum sizes in degraded situations in order to balance the cost of read against write operations within a partition. We have defined different reconciliation policies in order to re-establish replica and constraint consistency when nodes recover and network partitions rejoin.

Our availability analysis presented in [10] showed that AV provides better availability than traditional voting if (i) some of the data integrity constraints can be temporarily relaxed and (ii) reconciliation time is shorter than degradation time. Our experimental

evaluation indicates that AV is beneficial in configurations typical for our target applications.

References

[1] R.H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Trans. Database Syst.*, 4(2), 1979.

[2] D.K. Gifford. Weighted voting for replicated data. In *SOSP '79: Proc. of the 7th ACM Symp. on Operating Systems Principles*, pages 150–162. ACM Press, 1979.

[3] R. Smeikal and K.M. Goeschka. Fault-tolerance in a distributed management system: a case study. In *Proc. 25th Int. Conf. on Software Engineering*, pages 478–483. IEEE, 2003.

[4] Epics - experimental physics and industrial control system. http://aps.anl.gov/epics/.

[5] J. Osrael, L. Froihofer, K.M. Goeschka, S. Beyer, P. Galdámez, and F. Muñoz. A system architecture for enhanced availability of tightly coupled distributed systems. In *Proc. of 1st Int. Conf. on Availability, Reliability, and Security*. IEEE, 2006.

[6] F. Cristian. Understanding fault-tolerant distributed systems. *Commun. ACM*, 34(2), 1991.

[7] L. Froihofer, J. Osrael, and K.M. Goeschka. Trading integrity for availability by means of explicit runtime constraints. In *Proc. 30th Int. Computer Software and Applications Conference*. IEEE, 2006.

[8] M.J. Fischer, N.A. Lynch, and M.S. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of the ACM*, 32(2):374–382, 1985.

[9] S.L.K. Rountree and B.E. Gillet. Parametric integer linear programming: A synthesis of branch and bound with cutting planes. *European Journal of Operations Research*, 1982.

[10] J. Osrael, L. Froihofer, M. Gladt, and K.M. Goeschka. Availability of the adaptive voting replication protocol. Technical Report IR3.3-TUV-03, DeDiSys Consortium, 2006.

[11] P. Urban, X. Defago, and A. Schiper. Neko: a single environment to simulate and prototype distributed algorithms. In *Proc. 15th Int. Conf. on Information Networking*, pages 503–511. IEEE, 2001.

[12] S. Jajodia and D. Mutchler. Dynamic voting algorithms for maintaining the consistency of a replicated database. *ACM Trans. Database Syst.*, 15(2):230–280, 1990.

[13] J. Pâris. Voting with witnesses: A consistency scheme for replicated files. In *Proc. of the 6th Int. Conf. on Distributed Computing Systems*, pages 606–612. IEEE, 1986.

[14] P. Felber and P. Narasimhan. Reconciling replication and transactions for the end-to-end reliability of corba applications. In *Proc. of Confederated Int'l Conf. DOA, CoopIS and ODBASE 2002*, volume 2519 of *LNCS*, pages 737–754. Springer, 2002.

[15] Y. Ren, D.E. Bakken, T. Courtney, M. Cukier, D.A. Karr, P. Rubel, C. Sabnis, W.H. Sanders, R.E. Schantz, and M. Seri. Aqua: An adaptive architecture that provides dependable distributed objects. *IEEE Trans. on Computers*, 52(1):31–50, Jan. 2003.

[16] H. Yu and A. Vahdat. Design and evaluation of a conit-based continuous consistency model for replicated services. *ACM Trans. Comput. Syst.*, 20(3):239–282, 2002.

[17] D.B. Terry, M.M. Theimer, K. Petersen, A.J. Demers, M.J. Spreitzer, and C.H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. In *Proc. 15th ACM Symp. on Operating Systems Principles*, pages 172–182. ACM, 1995.

[18] J. Gray, P. Helland, P. O'Neil, and D. Shasha. The dangers of replication and a solution. In *Proc. Int. Conf. on Management of Data*, pages 173–182. ACM, 1996.

[19] N. Preguica, C. Baquero, F. Moura, J. Legatheaux Martins, R. Oliveira, H. Domingos, J. Pereira, and S. Duarte. Mobile transaction management in mobisap. In *Current Issues in Databases and Information Systems*, volume 1884 of *LNCS*, pages 379–386. Springer, 2000.

[20] Y. Saito and M. Shapiro. Optimistic replication. *ACM Comput. Surv.*, 37(1):42–81, 2005.