Availability/Consistency Balancing Replication Model

Johannes Osrael, Lorenz Froihofer, Karl M. Goeschka

Vienna University of Technology, Institute of Information Systems
Argentinierstrasse 8/184-1, 1040 Vienna, Austria
johannes.osrael|lorenz.froihofer|karl.goeschka@tuwien.ac.at

Abstract

Replication combined with explicit management of data integrity constraints can be used to enhance availability of object-oriented, data-centric distributed systems when node and link failures occur. Our approach enhances availability by temporarily relaxing non-critical data integrity constraints during degraded situations. This requires new kinds of optimistic replication protocols that support the configuration of this trade-off. The contribution of this paper is a replication model called Availability/Consistency Balancing Replication Model that allows replicas to diverge in degraded situations if data integrity can be temporarily relaxed and re-establishes both replica consistency and data integrity during repair time. The Primary-per-Partition-Protocol and Adaptive Voting are two concrete protocols following our model. The feasibility of our approach has been shown by several prototype implementations.

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.

Traditional replication protocols (partially) block in degraded situations (node and link failures) in order to guarantee that consistency is not violated. However, some applications exist where consistency can be temporarily relaxed in order to achieve higher availability. For instance, in some safety-critical systems (e.g., [1]) or in some control engineering applications (e.g., [2]) availability is more important than consistency.

New kinds of replication protocols are required that support the configuration of this trade-off between availability and consistency.

Thus, in this paper, we contribute with an enhanced replication model for balancing data integrity with availability, the *Availability/Consistency Balancing Replication Model* (ACBRM). The ACBRM is an extension of the model presented by Wiesmann et al. [3]. The Primary-per-Partition-Protocol [4] and Adaptive Voting [5] are two concrete protocols that follow our model.

Paper Overview: Our system model is presented in Sect. 2. Section 3 introduces the key concept of our novel replication protocols. Section 4 describes the replication model in detail. Section 5 presents two concrete protocols that realize the model: The Primary-per-Partition-Protocol and Adaptive Voting. Section 6 introduces our proof of concept implementations. Related work is discussed in Sect. 7 before we summarize and conclude in Sect. 8.

2 System Model

We focus on tightly-coupled, data-centric, object-oriented distributed systems with a small number of server nodes (typically 2-10) 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 consider both node and link failures (partitioning), i.e., 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. We assume that read and write operations on such objects can be distinguished.

Data integrity constraints: Not all constraints of an application are of equal importance [7]. Some have to be satisfied at any point in time while others might be relaxed temporarily when failures occur:

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.

Intra-object constraints can be evaluated on a single (logical) object, e.g. $object.attribute < constant$. *Inter-object* constraints need access to two or more objects, e.g. $object1.attr1 < object2.attr2$.

Critical operations affect at least one non-tradeable constraint while *non-critical* operations affect only tradeable constraints.

3 Key Concept

Traditional replication protocols (partially) block in degraded situations, e.g., if the primary is not reachable in a passive replication scheme or a quorum of replicas cannot be acquired in case of quorum consensus protocols. However, as discussed in section 1, some systems do not require strict data integrity at all times, i.e., constraint consistency can be temporarily relaxed during degraded situations.

Thus, our key idea is to enhance availability of traditional replication protocols by allowing non-critical operations in degraded situations in *all* partitions, even if replicas might diverge and data integrity constraints are possibly violated (threatened). Different *reconciliation policies* are required to re-establish replica and constraint consistency after nodes rejoin.

Our new replication protocols distinguish three modes of operation: normal mode, degraded mode, and reconciliation mode. The current mode of the replication protocol depends on the system state, as it is locally perceived by each node (see Fig. 1).
Our replication protocols are 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.

The protocols switch 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.

The protocols enter *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 consistency can only be re-established if all nodes are reachable. Thus, if partitions rejoin but the merged partition does not contain all nodes, either constraint consistency is re-established within the partition or constraint consistency is ignored and only replica consistency is re-established.

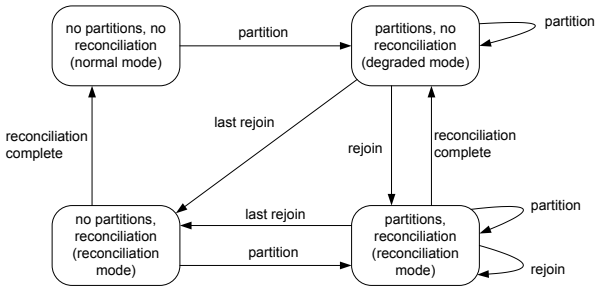

Figure 1. System states and protocol modes

4 Availability/Consistency Balancing Replication Model

Our contribution is a replication model called *Availability/Consistency Balancing Replication Model (ACBRM)* that extends the functional model for replication protocols introduced by Wiesmann et al. [3] with respect to the trading of constraint consistency against availability.

In Wiesmann et al's model, replication protocols are described as a sequence of five generic phases:

- `Request (RE):` The client submits an operation to one or more replicas.
- `Server coordination (SC):` The replicas coordinate with each other to synchronize the execution of the operation (ordering of concurrent operations).

- `Execution (EX):` The operation is executed on the replicas.
- `Agreement coordination (AC):` The replicas agree on the result of the execution (e.g., to guarantee atomicity).
- `Response (END):` The outcome of the operation is transmitted back to the client.

Some of the replication techniques skip one or more phases, order them in another way, iterate over some, or merge some of the phases.

We extend the model of Wiesmann et al. by introducing new phases that enable the balancing of data integrity with availability. The new phases are only required for non-critical operations except the constraint validation phase which is also required for critical operations. Beside this phase, critical operations are treated as in Wiesmann et al's model.

4.1 Normal Mode

In normal mode, replication protocols for trading availability against consistency behave similar as traditional replication protocols, with the only extension that data integrity constraints are explicitly supported and validation of the constraints is triggered. Thus a new phase, called *constraint validation (CV)*, is required.

4.1.1 Constraint Validation

The constraint validation phase starts immediately after the *execution (EX)* phase of a write operation. Constraint validation is not performed in case of read operations.

Figure 2. Protocol phases in normal mode for write operations

The write operation is aborted — independent of the type of the constraint (tradeable or non-tradeable) — if one of the constraints is not met since our protocols guarantee data integrity in the healthy system. The CV phase is identical for all our concrete protocols.

4.2 Degraded Mode

The key idea of our replication protocols is to allow non-critical operations in degraded situations in order to enhance availability, even if the consequence is that replicas might diverge and data integrity is threatened. Critical updates are treated as in normal mode.

Three new phases are introduced for the degraded mode of our protocols:

4.2.1 Configuration Adjustment (CA)

Replication protocols are configured with respect to various system parameters as the number of nodes, read to write ratio, load, etc. For example, in a quorum consensus scheme, the read and write quorums need to be configured. In a primary-backup approach, the roles (primary vs. backup) of the replicas have to be defined. Some replication protocols require reconfiguration in response to failures in order to provide fault tolerance, e.g., a new primary needs to be elected if the original primary crashes. In other protocols, as static quorum schemes, no intervention is necessary when failures occur, i.e., failures are masked.

Our protocols allow non-critical operations in all partitions during degraded situations. Thus, we adapt the protocols in degraded situations: For instance, in case of our Primary-per-Partition-Protocol (see section 5.1) we elect a temporary primary in each partition for objects that are only affected by tradeable constraints. Configuration adjustment is partition-internal and must be based on partition-specific parameters as the number of replicas or the roles of the replicas residing in the partition.

4.2.2 Constraint Validation (CV)

For critical operations, constraint validation is performed as in normal mode. However, for non-critical operations, which are allowed in different partitions, constraint validation has limited significance: A tradeable constraint that is satisfied based on the objects in the current partition might be violated retrospectively if one of the involved objects is changed in another partition. Thus, it can be configured whether or not constraint consistency within the partition shall be enforced. In the latter case, tradeable constraints do not have to be validated in degraded mode but are marked (in the reconciliation preparation phase) for validation at reconciliation time.

4.2.3 Reconciliation Preparation (RP)

In order to allow maximum flexibility for reconciliation when nodes rejoin, the replication protocol needs to log information about non-critical updates during degraded mode. In principle, either operations or states can be logged. Furthermore, depending on the reconciliation strategy, it is required to log either all, some,

or none of the operations and/or states. Logging everything (*full history*) offers all options during reconciliation but is the most resource-consuming approach. Keeping a *partial history* is a compromise between resource consumption and reconciliation flexibility and the third, *no history*, approach limits reconciliation to roll-forward or compensation actions that do not require a history of tentative operations/states.

Furthermore, the tradeable constraints affected by updates during degraded situations are marked for re-evaluation at reconciliation time.

Read operations do not require this phase: Read-write conflicts can be ignored during reconciliation since the application is aware that non-critical read operations performed during degraded situations return possibly stale objects.

Figure 3 depicts the sequence of the protocol phases in degraded mode for non-critical write operations. The configuration adjustment is the first phase and is triggered by the group membership service when the number of replicas in the current partition changes. However, not all changes require reconfiguration. For instance, this phase can be skipped in case of our primary-backup scheme if the original primary is still in the partition. If constraint consistency shall be enforced within the partition, constraint validation is performed immediately after the write operation is executed. Preparation for reconciliation starts afterwards.

Figure 3. Protocol phases in degraded mode for non-critical write operations

4.3 Reconciliation Mode

Reconciliation mode starts after the new group view is established because of (partial) partition re-unification. The overall goal of reconciliation is to re-establish constraint consistency which implies re-establishment of replica consistency. Full consistency (i.e., system-wide) can only be re-established if all nodes are available (full constraint consistency re-establishment). If this is not the case, we consider two options: Either constraint (and thus also replica) consistency is re-established within the partition (partial constraint consistency re-establishment) or only replica consistency is re-established (partial replica consistency re-establishment). Our concrete replication protocols that follow the ACBRM allow to plug-in different reconciliation policies.

4.3.1 Full/Partial Replica Consistency Re-establishment (RCR)

If updates have occurred in only one partition, the updates of this partition are applied on a certain number of replicas (depending on the concrete protocol) in the merged partition.

Replica conflict detection: Replica conflicts — caused by updates in different partitions — can be detected based on syntactic and/or semantic information. Syntactic approaches use information about when, where, and by whom operations have been submitted. Examples for syntactic techniques are version vectors [9], time-stamps [10], or precedence graphs [11]. Semantic techniques [12] exploit properties such as commutativity of idempotency of operations. Our replication protocols use only syntactic information (version numbers) for detection of replica conflicts since syntactic approaches provide better scalability than semantic ones. Thus, conflicts are detected by comparing the version histories of the partitions. How the version histories are generated depends on the concrete protocol.

Replica conflict resolution: If conflicts have been detected, one of the conflicting updates is chosen and the other updates are replayed or rolled back. Selection criteria are the number of updates, number of nodes in the partition, etc. Alternatively, a completely new or a default version can be installed to solve the conflict.

Constraint consistency re-establishment can only be performed if replica consistency is re-established beforehand. One option is to completely decouple replica and constraint consistency re-establishment to reduce complexity. Another option is to combine both phases: For instance, another selection criterion for replica conflicts is whether or not data integrity is satisfied by choosing one or the other version. If none of the replicas satisfies the constraints, one can be chosen according to the previously mentioned policies.

4.3.2 Full/Partial Constraint Consistency Re-establishment (CCR)

If at least one constraint is violated after re-establishment of replica consistency, the following policies can be distinguished to re-establish constraint consistency:

Re-schedule and replay: One option is to replay all tentative operations based on the last (replica and constraint) consistent state in a schedule that accepts as many operations as possible, under the conditions that both ordering and data integrity constraints are satisfied. This approach requires logging of all operations during degradation and suffers from scalability

problems.

Stepwise rollback: Another option is to stepwise revert objects affected by the violated constraint to previous versions till the constraint is satisfied. In the worst case, all operations are undone. This approach requires logging of all tentative states in degraded mode (i.e., full history approach).

Compensation actions: In order to avoid time-consuming rollbacks or replays, application-specific compensation actions can be defined for some applications. For instance, a simple compensation action is to choose a default or completely new (agreed) version in case of a conflict. This approach is even possible if no history is maintained.

The chosen version that satisfies the constraints is applied at a certain number of objects (depending on the concrete protocol) in the merged partition. If constraint consistency is re-established system-wide, the version histories are cleaned and constraint re-evaluation flags are set to false.

4.3.3 Configuration Adjustment (CA)

The configuration of our replication protocols is re-adjusted depending on the new situation after consistency is (partially) re-established. For instance, in our primary-backup scheme one of the two (temporary) primaries of the merged partition needs to be demoted to a secondary replica. In case of our adaptive voting protocol, the quorum sizes are adapted to the size of the partition.

Figure 4 depicts the sequence of the protocol phases in reconciliation mode. Replica consistency re-establishment is followed by constraint consistency re-establishment or might even be combined. Finally, the configuration of the protocol is adjusted (CA phase).

Figure 4. Protocol phases in reconciliation mode

4.4 Dependencies between Degraded and Reconciliation Mode

The number of reconciliation options depends on the reconciliation preparation phase as depicted in Fig. 5. Vice versa, reconciliation mode retrospectively influences the replication behavior in degraded mode. In case of the "reschedule/replay" and the "compensation actions" reconciliation approaches, the protocol behavior in degraded mode can be overwritten. For instance, it is even possible to retrospectively switch from a primary-backup based scheme to a voting algorithm. By applying a "stepwise rollback" reconciliation strategy, the effects of the degraded mode cannot be changed, though (partly) revoked.

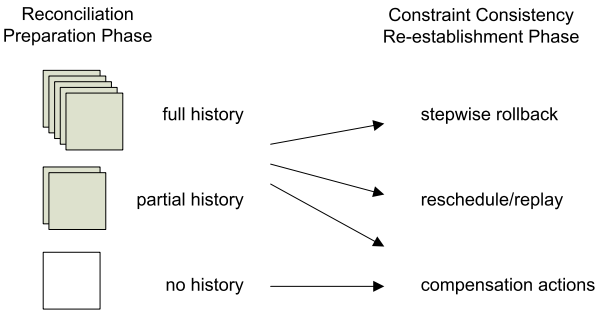

Figure 5. Dependencies between degraded and reconciliation mode

5 Concrete Protocol Examples

In this section we present two concrete replication protocols that realize our availability/consistency balancing replication model. One is based on the primary-backup approach while the other one builds upon quorum consensus. In principle, other protocols as coordinator-cohort [13] or active replication [14] can be adapted in a similar way.

5.1 Primary-per-Partition-Protocol

In contrast to the traditional primary partition approach [15], which allows only one partition to continue, a temporary primary is chosen if the original primary of an object (that is only affected by tradeable constraints) is not available in a partition. Hence we call it the *Primary-per-Partition-Protocol* (P4) [4].

Normal mode: Clients send their requests to the primary replica which executes the request, validates the constraints, and propagates the updates to the backups if the constraints are met. We use synchronous (eager) update propagation, i.e., a response is sent to the client after the replicas are updated.

Degraded mode: The P4 re-configures (CA phase) once failures are detected and the primary is not in the current partition, i.e., a new temporary primary is

elected for objects that are only affected by tradeable constraints. The tentative states are logged in the RP phase.

Reconciliation mode: Conflicts between concurrent updates in different partitions which are merged can be easily detected by comparing the version histories of the two temporary primaries (RCR phase). The P4 allows to plug-in different reconciliation protocols (e.g. [16]).

5.2 Adaptive Voting Protocol

We 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. Thus our adaptation of quorum consensus for balancing data integrity with availability is called *Adaptive Voting* (AV) [5].

Normal mode: In normal mode, AV behaves as the traditional voting protocol with the enhancement that invariant constraints are checked in case of write operations: Write operations are performed on a write quorum WQ of replicas and read operations on a read quorum RQ. The quorum conditions $RQ + WQ > N$ and $WQ > \frac{N}{2}$ must be met in order to prevent write-write and read-write conflicts. N is the number of nodes in the system, i.e., we assume all nodes have the same number of votes. Each node hosts a replica of an object.

Degraded mode: AV allows non-critical operations even if the quorums of the healthy system cannot be acquired. 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 in the CA phase. The tentative states are logged in the RP phase.

Reconciliation mode: In contrast to the P4, no individual node contains the full version history of a partition[1] since updates are performed on a write quorum which is smaller or equal than the number of nodes in the partition. Thus, in order to detect conflicting updates (RCR phase), the version history needs to be calculated based on the (partial) version histories of the nodes. As the P4, the Adaptive Voting protocol allows to plug-in several reconciliation strategies. The quorums are re-adjusted according to the size of the merged partition and the histories are cleaned up in the CA phase.

[1] Except a read-one/write-all scheme is applied in a partition.

6 Proof of concept implementations

6.1 Usage in Industrial Applications

The platform-independent system architecture of the DeDiSys replication middleware, which is targeted to these adaptive replication protocols for balancing data integrity with availability, has been presented in [5]. The DeDiSys middleware has been implemented on three different platforms: EJB [2], CORBA [17], and .NET [18]. The Primary-per-Partition-Protocol (P4), which is one of the concrete protocols that follows the ACBRM, has been implemented on all of these platforms. The Adaptive Voting (AV) protocol has been implemented for the .NET-based prototype [19].

Detailed test and validation reports of the DeDiSys middleware (including the replications protocols) and three different industrial applications (ATS (Alarm Tracking System [1]), ACS (Advanced Control System [20]), and EPICS (Experimental Physics and Industrial Control System [2]) Directory Service) that build upon DeDiSys can be found in [1, 2, 20].

Since reconciliation time is one of the key factors for overall availability of replication protocols that follow the ACBRM (see [19] for an availability analysis of Adaptive Voting) we provide some results from our experiments in the next subsection.

6.2 Performance of Reconciliation

The measurements were conducted on a 100MBit full duplex switched network with up to ten machines with similar strengths (1-3GHz, 1-3GB RAM, Windows Server 2003). Spread [21] has been used as group communication toolkit. For every experiment we performed three (independent) iterations of 1000 runs. In order to reduce the effects of just-in-time compilation (JIT), we performed 1000 runs before each iteration. The figure in this paper shows the average values over all iterations and runs.

We have evaluated the performance of Adaptive Voting using the following simple scenario: The state of an object a/b of class A/B is represented by an integer value x. Three constraints exist: C1: $a.x < constant1$, C2: $b.x < constant1$, C3: $a.x + b.x < constant2$. C1 and C2 are non-tradeable but C3 is tradeable.

We have measured (see Fig. 6) the worst and best case for reconciliation time in our scenario, depending on the number of operations applied in each par-

tition during degraded mode. Our system splits into two partitions containing three nodes each. A Majority scheme, i.e. Write Quorum = Read Quorum = 2, is applied in both partitions and constraints are enforced within the partitions. Object a is updated in partition 1, object b is updated in partition 2. The best case for reconciliation is if the two partitions can simply be merged without violating the inter-object constraint. However, if the inter-object constraint cannot be fulfilled by simply merging the partitions, one partition is stepwise rolled back till the constraint is fulfilled — in the worst case to the initial state before degradation occurred.

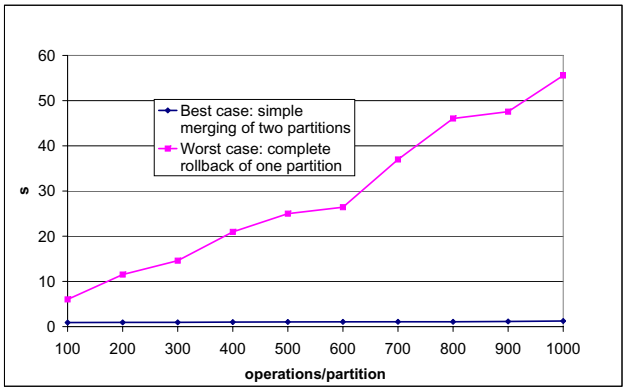

Figure 6. Example for reconciliation time

7 Related Work

Trading replica consistency for increased availability has been addressed in distributed object systems as [22, 23, 24]. However, these systems either guarantee strong replica consistency or no replica consistency at all. TACT (Tunable Availability and Consistency Trade-offs) [25] fills the space in 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 [26], 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. [27] 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.

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

8 Summary and Conclusion

We presented the Availability/Consistency Balancing Replication Model (ACBRM), an enhancement of Wiesmann et al's replication model [3] that enables the balancing of data integrity with availability in degraded situations when node and link failures occur. The key idea is to allow non-critical operations in degraded situations in all partitions even if replicas might diverge and data integrity constraints are possibly violated (threatened). We have defined different reconciliation policies in order to re-establish replica and constraint consistency when nodes recover and network partitions rejoin.

The Primary-per-Partition-Protocol [4] and Adaptive Voting [5] are two concrete protocols that follow the ACBRM. The feasibility of our approach has been shown by several prototype implementations ([28, 19, 17]).

9 Acknowledgements

This work has been partially funded by the European Community under the FP6 IST project DeDiSys (Dependable Distributed Systems, contract number 4152, www.dedisys.org). We thank Norbert Chlaupek for implementation and performance measurements of the Adaptive Voting replication protocol.

References

[1] H. Kuenig (ed.). FTNS/EJB. Technical Report D3.2.2, DeDiSys Consortium (www.dedisys.org), 2006.

[2] I. Habjan (ed.). FTNS/.NET. Technical Report D3.3.2, DeDiSys Consortium (www.dedisys.org), 2006.

[3] M. Wiesmann, F. Pedone, A. Schiper, B. Kemme, and G. Alonso. Understanding replication in databases and distributed systems. In *Proc. of 20th Int. Conf. on Distributed Computing Systems*. IEEE CS, 2000.

[4] Stefan Beyer, Mari-Carmen Bañuls, Pablo Galdámez, Johannes Osrael, and Francesc Muñoz. Increasing availability in a replicated partitionable distributed object system. In *Proc. 4th Int. Symp. on Parallel and Distr. Processing and Appl. (ISPA'06)*, volume 4330 of *LNCS*, pages 682–695. Springer, 2006.

[5] Johannes Osrael, Lorenz Froihofer, Matthias Gladt, and Karl M. Goeschka. Adaptive voting for balancing data integrity with availability. In *On the Move to Meaningful Internet Systems 2006: Confederated Int. Workshops Proc.*, volume 4278 of *LNCS*, pages 1510–1519. Springer, 2006.

[6] F. Cristian. Understanding fault-tolerant distributed systems. *Commun. ACM*, 34(2):56–78, 1991.

[7] Lorenz Froihofer, Johannes Osrael, and Karl M. Goeschka. Trading integrity for availability by means of explicit runtime constraints. In *Proc. 30th Int. Computer Software and Applications Conference*, pages 14–17. IEEE CS, 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] D.S. Parker Jr., G.J. Popek, G. Rudisin, A. Stoughton, B.J. Walker, E. Walton, J.M. Chow, D.A. Edwards, S. Kiser, and C.S. Kline. Detection of mutual inconsistency in distributed systems. *IEEE Trans. Software Eng.*, 9(3):240–247, 1983.

[10] S.K. Madria. Handling of mutual conflicts in distributed databases using timestamps. *The Computer Journal*, 41(6):376–385, 1998.

[11] S.B. Davidson. Optimism and consistency in partitioned distributed database systems. *ACM Trans. Database Syst.*, 9(3):456–481, 1984.

[12] A.-M. Kermarrec, A. Rowstron, M. Shapiro, and P. Druschel. The icecube approach to the reconciliation of divergent replicas. In *PODC '01: Proc. 20th ACM Symp. on Principles of Distributed Computing*, pages 210–218. ACM Press, 2001.

[13] K.P. Birman, T.A. Joseph, T. Raeuchle, and A. El Abbadi. Implementing fault-tolerant distributed objects. *IEEE Trans. on Software Engineering*, 11(6):502–508, 1985.

[14] F.B. Schneider. Replication management using the state-machine approach. In S.J. Mullender, editor, *Distributed Systems*, chapter 2, pages 17–26. ACM Press, Addison-Wesley, 2nd edition, 1993.

[15] A. Ricciardi, A. Schiper, and K. Birman. Understanding partitions and the "non partition" assumption. In *IEEE Proc. of 4th Workshop on Future Trends of Distributed Systems*. IEEE CS, 1993.

[16] Mikael Asplund and Simin Nadjm-Tehrani. Post-partition reconciliation protocols for maintaining consistency. In *Proc. Symposium on Applied computing*, pages 710–717. ACM Press, 2006.

[17] Stefan Beyer, Francesc D. Munoz-Escoi, and Pablo Galdamez. Corba replication support for fault-tolerance in a partitionable distributed system. In *Workshop Proc. of the 17th Int. Conf. on Database and Expert Systems Applications*, pages 406–412. IEEE CS, 2006.

[18] Johannes Osrael, Lorenz Froihofer, Georg Stoifl, Lucas Weigl, Klemen Zagar, Igor Habjan, and Karl M. Goeschka. Using replication to build highly available .NET applications. In *Workshop Proc. of the 17th Int. Conf. on Database and Expert Systems Applications*, pages 385–389. IEEE CS, 2006.

[19] Johannes Osrael, Lorenz Froihofer, Norbert Chlaupek, and Karl M. Goeschka. Availability and performance of the adaptive voting replication protocol. In *Proc. 2nd Int. Conf. on Availability, Reliability and Security*. IEEE CS, 2007.

[20] K. Zagar (ed.). FTNS/CORBA. Technical Report D3.4.2, DeDiSys Consortium (www.dedisys.org), 2006.

[21] J. Stanton Y. Amir, C. Danilov. A low latency, loss tolerant architecture and protocol for wide area group communication. In *Proc. of The Int. Conf. on Dependable Systems and Networks*, pages 327–336. IEEE CS, 2000.

[22] P. Felber and P. Narasimhan. Reconciling replication and transactions for the end-to-end reliability of corba applications. In *Proc. of OTM 2002*, volume 2519 of *LNCS*, pages 737–754. Springer, 2002.

[23] R. Guerraoui, P. Felber, B. Garbinato, and K. Mazouni. System support for object groups. In *OOPSLA '98: Proc. of the 13th ACM SIGPLAN Conf. on Object-oriented programming, systems, languages, and applications*, pages 244–258. ACM Press, 1998.

[24] 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.

[25] 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.

[26] 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 Press, 1995.

[27] 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 Press, 1996.

[28] Lorenz Froihofer, Gerhard Glos, Johannes Osrael, and Karl M. Goeschka. Overview and evaluation of constraint validation approaches in Java. In *Proc. 29th Int. Conf. on Software Engineering*. IEEE CS, 2007.