Availability and Performance of the Adaptive Voting Replication Protocol

Johannes Osrael[1], Lorenz Froihofer[1], Norbert Chlaupek[2], and Karl M. Goeschka[1]

[1]Vienna University of Technology, Argentinierstrasse 8/184-1, 1040 Vienna, Austria,

{johannes.osrael|lorenz.froihofer|karl.goeschka}@tuwien.ac.at

[2]University of Applied Sciences fh-campus wien, Daumegasse 5, 1100 Vienna, Austria, chlaupek@fh-campuswien.ac.at

Abstract

Replication is used to enhance availability and performance in distributed systems. Replica consistency and data integrity (constraint consistency) are correctness criteria for data-centric distributed systems. If consistency needs to be ensured all time, such systems soon become (partially) unavailable if node and link failures occur. However, some applications exist (e.g., in control engineering) where consistency can be temporarily relaxed during degradation in order to achieve higher availability. Recently we proposed Adaptive Voting (AV), a novel replication protocol based on traditional quorum consensus (voting) that allows the configuration of this trade-off. AV allows 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.

The contribution of this paper is two-fold: First, we present an availability analysis of AV. Second, we present performance measurements of AV from our .NET-based proof-of-concept implementation. The availability analysis shows that AV 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. The performance results indicate that a read-one/write-all configuration of AV is fastest for write-operations that involve checking of inter-object constraints and thus implicitly include read operations.

1 Introduction

Data-centric applications are considered as *constraint consistent* if all data integrity constraints are satisfied. Examples for data integrity constraints are value constraints, relationship constraints (cardinality, XOR), and uniqueness constraints. Replication, the process of maintaining different copies of an entity (e.g., object, data item), is the primary means to achieve high availability in a distributed system.

If strict constraint consistency has to be ensured all the time in a distributed system with replicated entities - even in the presence of failures - the system becomes (at least partially[1]) unavailable during degraded situations (node or link failures): Neither potentially conflicting updates on replicas in different partitions nor updates that possibly violate data integrity constraints are allowed.

However, some applications exist where consistency can be temporarily relaxed in order to achieve higher availability. Traditional replication protocols do not support the balancing between these two properties. Thus, we proposed a novel replication protocol called *Adaptive Voting* (AV) [3] 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.

Applications where AV can be used are for example the Alarm Tracking System (ATS) [4] and the Advanced Control System (ACS) [5]. The ACS will be

---

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

used as control system for the Atacama Large Millimeter Array radio telescope and is already widely used in the astronomy instrumentation control community. The ATS is based on a workflow management system and used for railway administration.

In this paper, our contributions are (i) an analysis of the availability of AV in comparison with traditional voting and (ii) an experimental evaluation of our .NET-based proof-of-concept implementation of AV.

Paper Overview Our system model is introduced in Sect. 2. Section 3 briefly discusses traditional voting, before we introduce the AV protocol in Sect. 4. Section 5 provides a comparison of the availability of traditional voting with AV. Section 6 presents performance measurements from our proof-of-concept implementation of AV. Our work is compared to related work in Sect. 7 before we conclude in Sect. 8.

2 System Model

We focus on tightly-coupled, data-centric, object-oriented distributed systems with up to about 30 server nodes[2] 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 Traditional 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:

$$RQ + WQ > V \tag{1}$$

$$WQ > \frac{V}{2} \tag{2}$$

$WQ, RQ, V \in \mathbb{N}$ 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.

4 Adaptive Voting

4.1 Key Idea

Traditional voting blocks operations if the quorums cannot be built. However, as discussed in Sect. 1,

---

[2]This figure stems from our real-world target applications [4, 5].

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

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 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) [3] allows to re-adjust the quorums in degraded situations in order to support the tuning 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. That is, re-establishment of replica consistency means in our protocol that a write quorum of replicas becomes consistent.

AV distinguishes three modes of operation: normal mode, degraded mode, and reconciliation mode. 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 (reconciliations) are finished. AV behaves as the traditional voting protocol with the enhancement that invariant constraints are checked in case of write operations. We denote the quorum sizes of the healthy system as WQ_H and RQ_H. I.e., in normal mode, write operations are performed on a write quorum WQ_H of replicas and read operations on a read quorum RQ_H. The quorum conditions $RQ_H + WQ_H > N$ and $WQ_H > \frac{N}{2}$ 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.

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 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. Tentative states are logged. Whether data integrity shall be enforced within a partition in degraded mode is configurable.

AV enters *reconciliation mode* when two or more partitions rejoin. The quorums are re-adjusted accord-

ing to the size of the merged partition. The objective of reconciliation is to re-establish replica and constraint consistency of the system. AV allows to plug-in different reconciliation policies, including strategies that (i) perform rollbacks in case of integrity violations, (ii) replay operations, or (iii) apply application-specific compensation actions, or (iv) combine these approaches. AV is specifically targeted to systems with the following characteristics:

- Most of the time all nodes are available and the system is in a healthy condition.

- Network partitions rarely happen. If they arise, the system typically splits into two parts. Nodes do not join/leave arbitrarily as in mobile, ad-hoc environments.

- Degradation is typically long-lasting. However, eventually, the nodes either rejoin or are (in case of a non-recoverable node crash) explicitly removed from the system, e.g., by maintenance intervention.

- Due to operational conditions (e.g., ownership of data), conflicts rarely happen.

4.2   Example

Figure 1 gives an example of the behaviour of AV in normal mode, degraded mode, and reconciliation mode.

The system consists of 5 nodes. The quorums are identical for all objects: $WQ_H = 4$, $RQ_H = 2$. Two objects, namely object A and object B are replicated. For the sake of simplicity, the state of the object is represented by an integer value. Furthermore, to enhance readability of the figure, we assume the integer value is always equal to the version number, i.e., a write operation can increment a or b by 1. The following tradeable inter-object constraint is defined: $A + B < 10$. This constraint has to be fulfilled in the healthy system (normal mode). For instance, A is updated 3 times and B is updated twice in the healthy system.

The system degrades into 2 partitions; none of the partitions contains a write quorum. The group membership service detects the failure and AV changes to degraded mode. Since the constraint is tradeable, operations are allowed in all partitions during degraded mode. Within a partition, update conflicts are avoided by applying a quorum scheme adapted to the size of the partition. Thus, the quorums are changed in both partitions. We denote the reduced quorums in partition k as WQ_k and RQ_k. The size of partition k is denoted as P_k in Fig. 1. For partition 1, both WQ_1
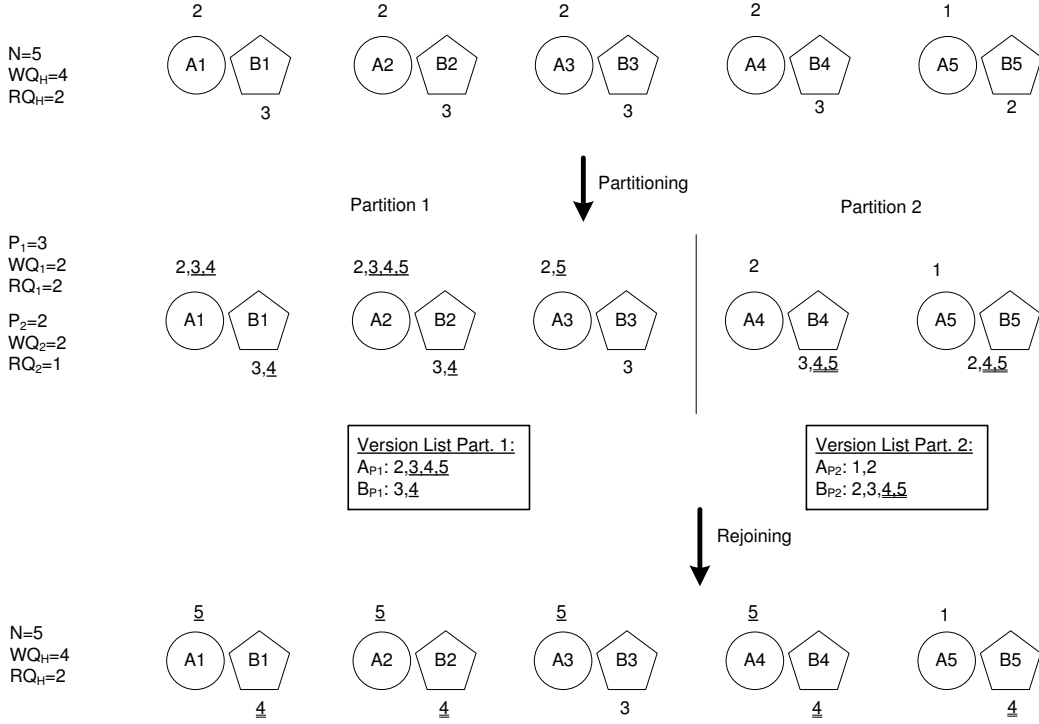
3

Figure 1. Example behaviour of AV

and RQ_1 are set to 2. In partition 1, A is set to 3 first, afterwards to 4, then to 5. Furthermore, B is set to 4 in partition 1. In partition 2, B is set to 4 first and then to 5. The states are logged in order to allow reconciliation at repair time.

Reconciliation starts when two or more partitions rejoin. Again, this is detected by the group membership service. The quorums are changed to the initial quorums (i.e., WQ_H and RQ_H) since all nodes are available. Afterwards, the version lists of the two partitions are compared. A has only been updated in partition 1, thus this version is chosen. B has been updated once in partition 1 and twice in partition 2. Version 4 of B is chosen (which has the same value in both partitions) since version 5 would violate the constraint. Thus, B is set to 4 on a write quorum WQ_H of nodes. All tentative versions are discarded. AV returns to normal mode.

5 Availability Analysis

Jiménez-Peris et al. [9] compare various quorum schemes with the conventional read-one/write-all-available (ROWAA) approach in terms of availability, scalability, and performance. Regarding availability, they conclude that ROWAA is the best choice for a

wide range of applications if no network partitions occur. However, if partitions are considered as in our target applications, ROWAA needs to adopt the primary partition approach and thus exhibits the same availability as majority voting [9]. Since majority voting is a special configuration of traditional voting, we compare availability of Adaptive Voting with availability of traditional voting (TV). For both protocols, availability of read and write operations needs to be distinguished. Total availability is expressed as

$$A = q_w A_w + q_r A_r \qquad (3)$$

A_w is the write availability of TV and A_r the read availability for TV. Availability of AV follows the same scheme. q_w and q_r express the weight of write and read operations. E.g., $q_w = 0.4$ denotes that 40% of all operations are write operations. Thus, $q_w + q_r = 1$ must hold. Equal load on all nodes is assumed.

5.1 Traditional Voting

For TV, write/read availability is the sum of the availabilities in each partition. P_i is the size of partition i, N is the total number of nodes. One replica per node is assumed. Write operations can only be performed in at most one partition, while read operations might be

4

performed in several partitions if no partition with a write quorum exists (see [3]). Thus, the values of A_w and A_r in this case are as follows:

$$A_w = \begin{cases} \frac{\max\limits_i P_i}{N} & : & \exists\, P_i \geq WQ_H \\ 0 & : & else \end{cases} \quad (4)$$

$$A_r = \sum_i \frac{P_i}{N} \quad \forall P_i \geq RQ \quad (5)$$

5.2 Adaptive Voting

For AV, non-critical and critical write operations need to be distinguished. The latter type has the same availability (denoted as $\overline{A}_{w_{critical}}$) as write operations of TV. Availability of the first type — denoted as $\overline{A}_{w_{non-critical}}$ — is 100% in normal mode and in degraded mode while such operations are not allowed in reconciliation mode.

$$\overline{A}_{w_{critical}} = A_w \quad (6)$$

$$\overline{A}_{w_{non-critical}} = \begin{cases} 1 & : & normal\ mode,\ degr.\ mode \\ 0 & : & reconciliation\ mode \end{cases} \quad (7)$$

Thus, total availability of write operations of AV is expressed as

$$\overline{A}_w = q_c\, \overline{A}_{w_{critical}} + q_{nc}\, \overline{A}_{w_{non-critical}} \quad (8)$$

where $q_c + q_{nc} = 1$ must hold. q_{nc} is the percentage of non-critical write operations while q_c is the percentage of critical write operations.

Read operations follow a similar scheme: Read operations on objects affected by non-tradeable constraints have the same availability as read operations in TV. Read operations on objects affected by tradeable constraints have 100% availability in normal mode and degraded mode but are blocked during reconciliation mode.

5.3 Availability over Time

So far, we have analyzed the availability of the system in several system states, depending on the number and sizes of partitions. The analysis shows that AV provides higher (or at least equal if all constraints are non-tradeable) availability than TV in degraded mode. However, availability of AV declines during reconciliation. Thus, in order to decide when AV is beneficial, availability needs to be considered over time. We denote each t_i as a point in time where one or more nodes

leave or rejoin. t_{ds} is the point in time where degradation starts and t_{re} where reconciliation ends. Availability is 100% in the healthy system for both protocols; thus, AV advances TV if

$$\frac{\sum\limits_{i=ds}^{re-1} \overline{A}(t_i) \cdot (t_{i+1} - t_i)}{t_{re} - t_{ds}} > \frac{\sum\limits_{i=ds}^{re-1} A(t_i) \cdot (t_{i+1} - t_i)}{t_{re} - t_{ds}} \quad (9)$$

holds. I.e., AV yields better availability over time if reconciliation time is short in comparison to degradation time.

Figure 2 shows how availability might change over time. The figure is an example that has been chosen for presentation purposes but does not represent real measurements.
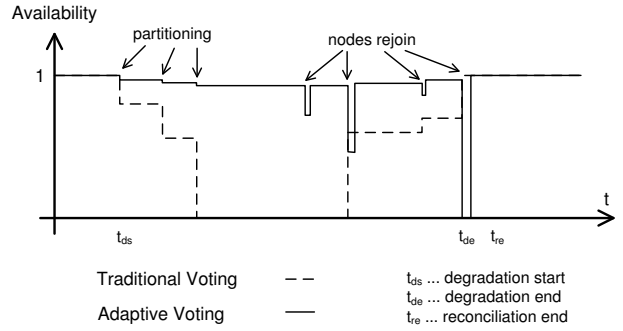

Figure 2. Availability over time: AV vs. TV

5.4 Influence of the Reconciliation Mode on Availability

Update conflicts and data integrity violations, which might be introduced in degraded mode since AV allows non-critical updates in all partitions, need to be resolved in reconciliation mode. Applying a rollback to a consistent checkpoint would revoke two types of operations. The first type would have been rejected in normal (healthy) mode anyway and thus their revocation would not retrospectively reduce availability. However, the second type of operations could have been applied successfully in normal mode either (i) as they are or (ii) as another operation according to the user's intention but based on the healthy context. If we do not want to reduce availability retrospectively at all, we have to assume that all update conflicts for the latter type of operations can be resolved by (i) replaying some operations or (ii) application-specific compensation actions. This is what we assume for our availability analysis.

The first type of operations, however, can simply be revoked/undone. Putting it the other way round: availability is only reduced retrospectively, if reconciliation revokes operations that could have been reasonably applied in the normal (healthy) mode. However, applying heuristics to cope with reconciliation complexity may further reduce availability retrospectively, which has to be investigated in future work.

6 Performance Evaluation

We have implemented AV in the .NET version [10] of our DeDiSys [11] replication middleware. DeDiSys is targeted to partitioned environments and allows to plug-in replication protocols for balancing data integrity against availability.

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 [12] 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. All figures in this paper show 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.

6.1 Latency of Operations

Normal mode: Figure 3 compares the latency of (i) a write operation with constraint checking[4], (ii) a write operation without constraint checking and (iii) a read operation in normal mode for different node numbers. Adaptive Voting is configured with a read-one/write-all (ROWA) strategy in this figure, i.e. $WQ = number\ of\ nodes$, $RQ = 1$. AV applies a write operation locally first and then propagates the whole object state of around one kilobyte to a write quorum of replicas. Write operations with constraint checking are slightly slower than write operations without constraint checking since they involve read operations as

---

[4]cc = constraint checking

well. However, in case of ROWA, the difference is negligible since the reads can be performed locally: a local read requires about five milliseconds. Latency of write operations directly depends on the number of nodes, while read operations can always be performed locally in an ROWA scheme.
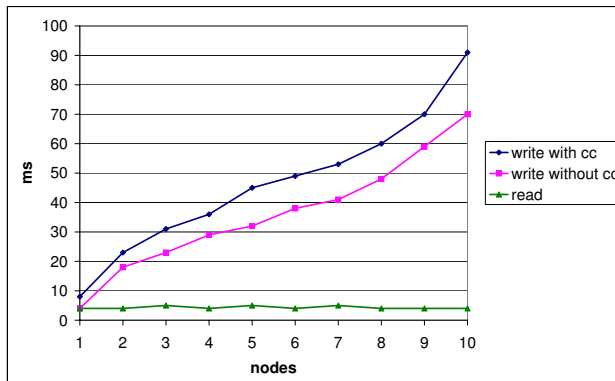

Figure 3. ROWA AV in normal mode

Figure 4 shows the latency of these operations for Majority Adaptive Voting, i.e. $WQ = \left\lfloor \frac{N}{2} \right\rfloor + 1$ and $RQ = \left\lceil \frac{N}{2} \right\rceil$. Read and write operations without constraint checking have similar performance since the quorums are either identical (for an uneven number of nodes) or differ only by one (for an even number of nodes). Write operations with constraint checking are much slower since they involve remote read operations as well due to the inter-object constraint C3.
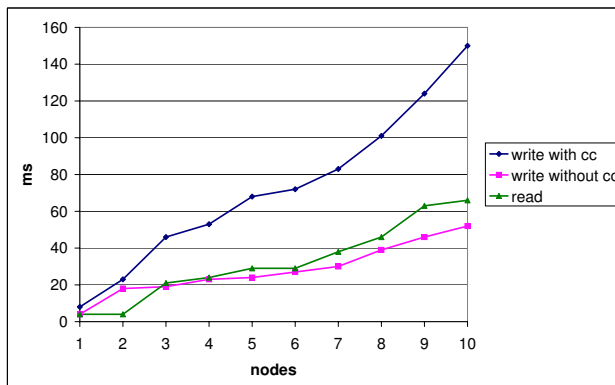

Figure 4. Majority AV in normal mode

ROWA and Majority are the two extreme configurations of the Adaptive Voting protocol. While ROWA performs best for read operations, Majority is faster for write operations without constraint checking. Whether the one or the other strategy is better for write op-

erations with constraint checking depends if information from remote nodes is required. In our example, which involves checking the inter-object constraint C3, ROWA would be the better strategy in terms of performance.

Besides these two extreme configurations, Adaptive Voting can be configured for other quorum sizes as well. Figure 5 compares the latency of operations for the possible quorum sizes if the number of nodes is ten:
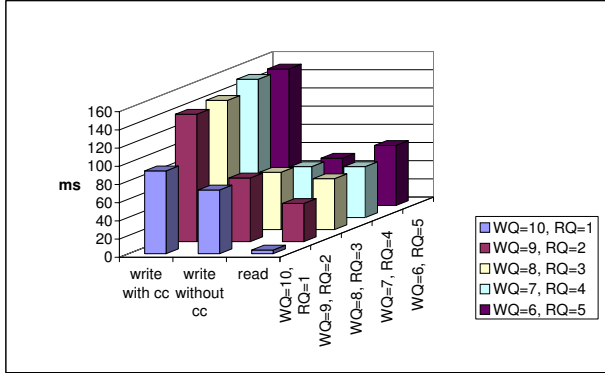

Figure 5. Quorum strategies for 10 nodes

In our example with an inter-object constraint, write operations with constraint checking perform best for the ROWA strategy since read operations required for constraint checking can be performed locally. The performance of write operations (without constraint checking) becomes slightly better with decreasing WQ. The trend for read operations is similar with decreasing RQ. I.e., the figure shows that the performance of read and write operations (without constraint checking) can be balanced against each other.

Degraded mode: Read operations and critical write operations have similar performance in degraded mode and normal mode for a comparable number of nodes. Non-critical write operations are slightly slower (for a comparable number of nodes) in degraded mode since logging of the operations and/or states is required.

6.2 Performance of Reconciliation

We have measured (see Figure 6) the worst and best case for the previously used simple scenario. Our system splits into two partitions containing three nodes each. An ROWA scheme 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.
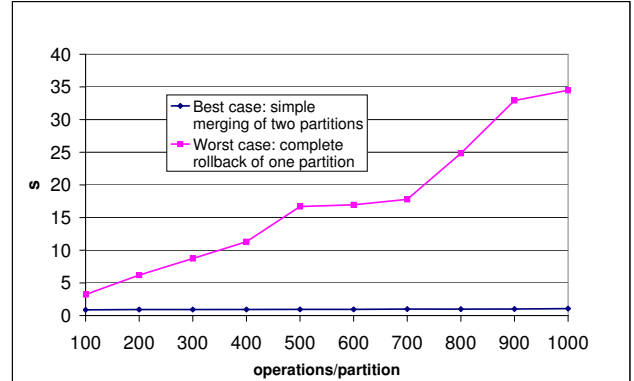

Figure 6. Example for reconciliation time

Performance of reconciliation is highly application-specific, i.e., it depends on the constraints, failure pattern, load during degradation, reconciliation policies, etc.

7 Related Work

Various dynamic quorum schemes (e.g., [13, 14]) have been proposed which adapt to changes in the system due to failures. However, in contrast to Adaptive Voting 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 [15, 16]. TACT [17] provides 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. Furthermore, different solutions for reconciliation of divergent replicas have been proposed for mobile environments (e.g., [18]).

All of the above replication and reconciliation approaches have one commonality: In contrast to our Adaptive Voting approach, they either do not address constraint consistency explicitly or presume strong data integrity.

The availability and performance of Adaptive Voting has not been analyzed in previous work. However, an excellent comparison of other voting schemes is presented in [9].

8 Conclusions

Adaptive Voting [3] is a novel 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 Adaptive Voting is to allow non-critical operations (that only affect tradeable data integrity constraints) even if the quorum conditions cannot be met. Different reconciliation policies have been defined to re-establish consistency during reconciliation.

The availability analysis presented in this paper shows that Adaptive Voting 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. The performance measurements presented in this paper give an indication for the choice of the quorum strategy in terms of performance. As in traditional voting, a read-one/write-all strategy is best for read-intensive applications while a Majority strategy is better for write-intensive applications. However, implicit read operations caused by constraint checking during a write operation (e.g., in case of inter-object constraints) have to be taken into account.

9 Acknowledgements

References

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

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

[3] J. Osrael, L. Froihofer, M. Gladt, and K.M. Goeschka. Adaptive voting for balancing data integrity with availability. In *OTM Confed. Int. Workshops Proc.*, volume 4278 of *LNCS*, pages 1510–1519. Springer, 2006.

[4] H. Kuenig (ed.). FTNS/EJB - software prototype & refined design & validation report. Technical Report D3.2.2, DeDiSys Consortium (www.dedisys.org), 2006.

[5] K. Zagar. Fault tolerance scenarios in control engineering. In P. Cunningham and M. Cunningham, editors, *Innovation and the Knowledge Economy - Issues, Applications, Case Studies*, volume 2, pages 1389–1395. IOS Press, 2005.

[6] F. Cristian. Understanding fault-tolerant distributed systems. *Commun. ACM*, 34(2), 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 (COMPSAC'06)*, 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] R. Jiménez-Peris, M. Patiño-Martínez, G. Alonso, and B. Kemme. Are quorums an alternative for data replication? *ACM Trans. DB Syst.*, 28(3):257–294, 2003.

[10] 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.

[11] J. Osrael, L. Froihofer, K. M. Goeschka, S. Beyer, P. Galdamez, and F. Munoz. A system architecture for enhanced availability of tightly coupled distributed systems. In *Proc. 1st Int. Conf. on Availability, Reliability and Security*, pages 400–407. IEEE CS, 2006.

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

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

[14] 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.

[15] 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*, pages 737–754. Springer, 2002.

[16] 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.

[17] 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.

[18] 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 Symp. on Operating Systems Principles*, pages 172–182. ACM Press, 1995.