The Management of Replicated Data

Jehan-François Pâris *

Department of Computer Science, University of Houston
Houston, TX 77204-3475

1 Introduction

Fourteen years have passed since Gifford's seminal paper on weighted voting [7]. These years have seen the development of numerous protocols for managing replicated data and a handful of experimental systems implementing replicated files. The time has now come to attempt an inventory of the problems for which we have found solutions and the issues that remain open. One way to structure this inventory is to organize it around general observations reflecting points of agreement and disagreement within the replicated data community.

Our frame of reference will be simple: We will consider systems maintaining multiple copies–or *replicas*–of the same data at distinct nodes of a computer network. We define the *availability* of replicated data for a given operation as the probability that the operation can be successfully carried out at some node within the network. We will focus on the problem of protecting the users of the replicated data from the inconsistencies that may result from node failures and network partitions. This is normally done through a *group communication mechanism* [3, 4, 22] or a *replication control protocol*. Group communication mechanisms focus on the problem of reliable delivery of messages to the replicas while replication control protocols operate by mediating all accesses to the replicated data. Hence they are more general. An ideal *replication control protocol* should guarantee the consistency of the replicated data in the presence of any arbitrary combination of non-Byzantine failures while providing the highest possible data availability and occasioning the lowest possible overhead.

2 What we have learned

Observation 1. *Applications that can tolerate slight inconsistencies among replicas should not have to pay the price of maintaining all replicas in a fully consistent state.*

Maintaining replicated data in a fully consistent state is an expensive proposition because all updates must become simultaneously visible to *all* users of the replicated data. Reducing the number of replicas required to obtain a write quorum cannot be achieved without simultaneously increasing the read quorum and vice versa.

* Internet Address: paris@cs.uh.edu

Many practical applications can tolerate minor inconsistencies among replicas and operate with data that are slightly out of date. We can provide these applications with replicas that are slightly out of date and even guarantee that these *quasi-copies* will never deviate too far from the true data [1]. For instance, we can guarantee that the quasi-copies will never be more than 10 minutes behind the true data or differ by more than 5%.

Other distributed applications only require *eventual consistency* [10]. Replicas can be in different states but will eventually receive the same set of update messages. Consider, for instance, a mail system where each user is given two mailboxes and incoming messages are guaranteed to be delivered to at least one mailbox. Merging the two mailboxes can be done very easily by sorting their contents according to message arrival time and removing duplicate messages.

Observation 2. *Network partitions are much more difficult to handle than node failures.*

In the absence of network partitions, we can guarantee the consistency of replicated data by (a) imposing a total ordering on all writes so that all replicas receive them in the same order, (b) broadcasting these writes to all available replicas, and (c) requiring that replicas residing on nodes recovering from a failure remain unavailable or *comatose* until they are brought up to date. This *available copies* protocol has two major advantages [2, 17]. First, read requests never need to access more than one available replica because *all* available replicas are guaranteed to be up to date. Second, the replicated data can be accessed as long as there is at least *one* available replica.

The situation is quite different when network partitions must be taken into account. A first class of replication control protocols takes the approach that network partitions are unlikely to occasion conflicting updates and that many of these conflicts will be easy to resolve. These are known as *optimistic protocols*. They follow the same philosophy as the available copies protocol and trade data consistency for inexpensive reads and high availability. The second class of protocols take the approach that data consistency carries a much more important weight than data availability. These protocols are said to be *pessimistic*. They rely on quorum mechanisms to prevent conflicting updates and provide lower data availabilities than optimistic protocols. For instance, we need *five* replicas managed by the best pessimistic protocol to achieve the same level of data availability as *two* replicas managed by the available copy protocol [17].

Another limitation of pessimistic protocols is the fact that they disallow simultaneous updates in disjoint partitions. Consider for instance a replicated file system like Coda [21] where each user workstation maintains a local copy of the files it currently accesses in addition to the copies maintained by the Coda servers. Some of these workstations are likely to be notebooks and we may expect the owners of these notebooks to disconnect them from time to time from the network. A pessimistic protocol would either disallow all writes to the Coda files stored on the notebook or disallow all accesses to the replicas maintained by the Coda servers.

Observation 3. *Witnesses can reduce the storage costs of pessimistic protocols.*

Pessimistic protocols require at least three replicas to implement a robust consensus and improve upon the availability of unreplicated data. However one of these replicas can be replaced by a much smaller *witness* without significantly affecting the availability of the replicated data [16].

Witnesses are very small entities that hold no data but maintain enough information to identify the replicas that contain what it believes to be the most recent version of the data. Conceptually this information could be a *timestamp* containing the time of the latest update. Since it is quite hard to keep clocks synchronized, this timestamp is normally replaced by a *version number*, which is an integer incremented each time the data are updated. Each witness carries a specific number of votes and is allowed to participate in all read and write quorums like a conventional replica.

The small size of witnesses offers two additional advantages. First, witnesses, unlike conventional replicas, can be brought up to date without any significant delay. There is thus no incentive to update witnesses that are not part of a write quorum. As a result, a replicated file consisting of two replicas and one witness will never require more than two replica updates per write while a replicated file consisting of three conventional replica normally requires three replica updates per write. Second, witnesses can be quickly regenerated every time they become unavailable [20, 19].

Witnesses are an integral part of the Echo [11] and Harp [14] file systems.

Observation 4. *Dynamic voting can improve the availability of replicated data managed by quorum-consensus protocols.*

Dynamic voting protocols adjust read and write quorums whenever they detect a change in the number of available replicas [5, 12]. Central to all dynamic voting protocols, is the notion that replicas known to be unreachable should be excluded from all quorum computations. All dynamic voting protocols maintain some record of the set of replicas that are allowed to participate in elections. Because of the distributed nature of the protocol, multiple copies of this set, called the *majority block*, must be maintained. These copies are normally associated with the replicas. Whenever the protocol detects that some replicas in the majority block have become unreachable it checks first that a majority of the replicas in the current majority block can be reached. If this is the case, the unreachable replicas are excluded from the majority block and a new majority block is formed. Otherwise the replicated data remain unavailable until some of the unreachable replicas can be reached again. Finally, the excluded replicas are prevented from participating in voting until they are formally reintegrated into the current majority block.

It has long been known that the best dynamic voting protocol, namely *dynamic-linear voting* [12], provided much better data availabilities than the best static voting protocols whenever there were more than three replicas. It was also widely assumed that dynamic-linear voting did not perform much

better than static voting when there were only three replicas. This author has shown more recently that this conclusion does not hold when communication failures are taken into account as static voting provides lower availabilities than dynamic-linear voting even when there are only three replicas [18]. Hence the small overhead of maintaining majority block membership information on each node holding a replica is a very reasonable price to pay for the much better data availabilities afforded by dynamic-linear voting.

Observation 5. *There are protocols that can manage efficiently large numbers of replicas.*

The recent years have seen the development of several replication control protocols specially tailored for the management of replicated data that have many replicas. These protocols have as objective to reduce the number of replicas that need to be accessed to reach a read or a write quorum while distributing these accesses as evenly as possible among the N replicas.

We shall only mention Maekawa's original algorithm [15] because it was the first algorithm to require only $3\sqrt{N}$ messages per access and the *triangular lattice* protocol because it provides a much better data availability than Maekawa's algorithm while keeping the quorum size of $O(\sqrt{N})$ [23].

There are also some wide-area applications, such as the *Archie* FTP location service [8] or the *Refdbms* bibliographic database system [9] , whose users can be distributed at hundreds or thousands of sites around the world. These applications often maintain a very large number of local replicas in order to provide a fast response time and minimize communication costs. Even protocols with $O(\sqrt{N})$ quorums would be too expensive. The only solution is then to relax consistency requirements. Less costly protocols, among which *epidemic* protocols [6, 10], can then be used.

3 Unresolved Issues

Observation 6. *We lack a proper consistency model providing for disconnected—or quasi-disconnected—operation of user workstations.*

Conventional network file systems assume that user workstations are permanently connected to their servers. This assumption is becoming false because of the increasing importance of portable workstations. These portable workstations have enough secondary storage to be fully autonomous and are equally likely to be operated in stand alone mode as to be connected to the network. As we mentioned earlier, pessimistic replication protocols are inadequate because they would unduly restrict the access of data. The Coda file system solves the problem by implementing an optimistic replication control protocol and guaranteeing that the user always sees the most recent accessible version of its data [21]. This solution has the disadvantage of shifting too much burden on the users' shoulders as they become at least partially responsible for the

consistency of their data. We need to develop consistency models that provide the users with a more faithful abstraction of the way the replicas are actually managed. We need to take also into account the emergence of new technologies providing portable workstations with more or less reliable low-bandwidth radio links. These links could be used for the exchange of tokens, for the update of witnesses or for the transmission of incremental updates.

Observation 7. *We need to develop better methodologies for specifying weak consistency criteria and implementing them.*

Weak consistency protocols allow replicas to diverge temporarily from one another but guarantee that they will eventually reach a single consistent state. Weak consistency protocols incur much lower communication overheads than conventional replication control protocols. They constitute the only practical way to manage very large numbers of replicas scattered over a wide-area network. Unfortunately managing weakly consistent data is a difficult task because it is very data dependent. Hence object-oriented methodologies appear to be the most promising approach [10].

Observation 8. *We need to develop tools measuring more accurately the actual performance of replication control protocols.*

Too many studies of replication control protocols still neglect network partitions and assume a perfect coverage of all node failures. These studies provide overoptimistic evaluations of the actual availabilities of the replicated data. As we mentioned earlier, they also fail to notice some behaviors that only occur in the presence of network partitions. The fault-tolerant computing community has an important part to play because of its impressive collective expertise in reliability and availability analysis.

4 Final Remarks

This brief inventory of the current state of the art in the field of replicated data management has neglected many interesting problems, among which the optimal allocation of weights to replicas, and failed to discuss many good protocols such as the tree protocol, hierarchical voting, and voting with ghosts to mention only a few ones.

We have also failed to mention the ever growing difference between replicated files and replicated databases. A replicated file is normally a relatively small object often under the control of a single user. As it is in the case for unreplicated files, this owner is quite likely to place a higher priority on faster access times and higher data availability than on data consistency. Hence optimistic replication control protocols are indicated. Replicated databases, on the other hand, need to rely on some formal model of data consistency because they are typically accessed in parallel by many users. As a result, even unreplicated databases require a formal transaction mechanism to guarantee that all updates will leave the data base in a consistent state. Hence temporary inconsistencies can only be tolerated if there are formal mechanisms to reconcile them.

References

1. Alonso, R., Barbara, D., Garcia-Molina, H.: Quasi-copies: efficient data sharing for information retrieval systems. Proc. of the Int. Conf. on Extending Data Base Technology, Lecture Notes in Computer Science # 303, Springer Verlag (1988).
2. Bernstein, P.A., Goodman, N.: An Algorithm for concurrency control and recovery in replicated distributed databases. ACM Trans. on Database Systems, **9**, 4 (1984) 596–615.
3. Birman, K.P., Joseph, T.A.: Reliable communication in the presence of failures. ACM Trans. on Computer Systems, **5**, 1 (1987) 47–76.
4. Cheriton, D.R., Zwaenepoel, W.: Distributed process groups in the V kernel. ACM Trans. on Computer Systems, **3**, 2 (1985) 77–107.
5. Davcev, D., Burkhard, W.A.: Consistency and recovery control for replicated files. Proc. 10th ACM Symp. on Operating System Principles, (1985) pp. 87–96.
6. Demers, A., Greene, D., Hauser, C., Irish, W., Larson, J., Shenker, S., Sturgis, H., Swinehart, D., Terry, D.: Epidemic algorithms for replicated database maintenance. Operating Systems Review, **22**, 1, (1988) 8–32.
7. Gifford, D. K.: Weighted voting for replicated data. Proc. 7th ACM Symp. on Operating System Principles, (1979) pp. 150–161.
8. Emtage, A., Deutsch, P.: Archie, an electronic directory service for the Internet. Proc. 1992 Winter USENIX Conf., (1992) pp. 93–110
9. Golding, R.A.: Weak-consistency group communication and membership. Ph.D. thesis published as Technical Report UCSC–CRL–92–52, Computer and Information Sciences Board, University of California, Santa Cruz (1992).
10. Golding, R.A.: A Weak-consistency architecture for distributed information services. Computing Systems, **5**, 4 (1992).
11. Hisgen, A., Birrell, A., Mann, T., Schroeder, M., Swart, G.: Availability and consistency tradeoffs in the Echo distributed file system. Proc. 2nd Workshop on Workstation Operating Systems, (1989) pp. 49–54.
12. Jajodia, S., Mutchler, D.: Dynamic voting algorithms for maintaining the consistency of a replicated database. ACM Trans. on Database Systems, **15**, 2 (1990) 230–405.
13. Ladin, R., Liskov, B., Shrira, L.: Lazy replication: exploiting the semantics of distributed services. Proc. 9th ACM Symp. on the Principles of Distributed Computing, (1990).
14. Liskov, B., Ghemawat, S., Gruber, R., Johnson, P., Shrira, L. Williams, M.: Replication in the Harp file system. Proc. 13th ACM Symp. on Operating System Principles, (1991) pp. 226–238.
15. Maekawa, M.: A $\sqrt{N}$ algorithm for mutual exclusion in decentralized systems. ACM Trans. on Computer Systems, **3**, 2 (1985) 145–159.
16. Pâris, J.-F.: Voting with witnesses: a consistency scheme for replicated files. Proc. 6th Int. Conf. on Distributed Computing Systems, (1986) pp. 606–612.
17. Pâris, J.-F., Long, D.D.E.: On the performance of available copy protocols. Performance Evaluation, **11** (1990) 9–30.
18. Pâris, J.-F.: Evaluating the impact of network partitions on replicated data availability. In Dependable Computing for Critical Applications 2 (J.F. Meyer and R.S. Schlichting eds.), Dependable Computing and Fault-Tolerant Systems #6, Springer Verlag (1992), pp. 49–65.
19. Pâris, J.-F., Long, D.D.E.: Voting with regenerable volatile witnesses. Proc. 7th Int. Conf. on Data Engineering, (1991) 112–119.

20. Pu, C., Noe, J.D., Proudfoot, A.B.: Regeneration of replicated objects, a technique and its Eden implementation. IEEE Trans. on Software Engineering, **SE-14**, 7 (1988) 936–945.
21. Satyanarayanan, M., Kistler, J.J., Kumar, P., Okasaki, M.E. Siegel, E.H., Steere, D.C.: Coda: a highly available file system for a workstation environment. IEEE Trans. on Computers, **C-39**, 4 (1990) 447–459.
22. Schneider, F. B.: Implementing fault-tolerant services using the state machine approach: a tutorial. ACM Computing Surveys, **22**, 4 (1990) 229–319.
23. Wu, C., Belford, G.: The Triangular lattice protocol: a highly fault tolerant and highly efficient protocol for replicated data. Proc. 11th Symp. on Reliable Distributed Systems, (1992) 66–73.