State-Machine Replication with Infinitely Many Processes*

Position Paper

Gregory Chockler, Dahlia Malkhi and Danny Dolev
School of Computer Science and Engineering
The Hebrew University of Jerusalem
Jerusalem, Israel 91904
{grishac,dalia,dolev}@cs.huji.ac.il

April 30, 2002

1 Introduction

The State Machine Replication (SMR) approach [14, 23] is a fundamental paradigm for implementing fault-tolerant distributed systems. In this approach, the service state is replicated at several sites and the operations which access (and possibly modify) the service state are delivered in the same total order without gaps at each site thus keeping the replica states consistent. The Paxos algorithm of Lamport [15, 17, 16, 8] is one of the most widely used techniques for implementing operation ordering for SMR. Numerous flavors of Paxos which adapt it for various settings and environments have been described in the literature, e.g., [13, 7, 4, 10].

At the core of Paxos is a Consensus algorithm called *Synod*. Since Consensus is unsolvable in asynchronous systems with failures [9], the Synod protocol, while guaranteeing always to be safe, ensures progress only when the system is stable so that an accurate leader election is possible. In order to guarantee safety even during instability periods, the Synod algorithm employs a 3-phase commit like protocol, where unique *ballots* are used to prevent multiple leaders from committing possibly inconsistent values, and to safely choose a possible decision value during the recovery phase.

Most of the existing Paxos implementations were designed for pure message passing environments, where the replicas in addition to being data holders, also actively participate in the ordering protocol. A natural alternative to this model would be to consider the Paxos implementation in shared memory-like systems. Here, the replicas are just persistent memory modules intended for coordination and information sharing among transient clients whose number and identities are not known in advance. This model faithfully represents several realistic application settings. The following examples illustrate this point:

Storage Area Networks : In Storage Area Networks (SANs), disks are directly attached to high speed networks that are accessible to clients. The SAN technology enables cost-effective bandwidth scaling by allowing data to be transferred directly from network attached disks to clients so that the file server bottleneck is eliminated. Since clients (or a group of designated SAN servers) need to coordinate and secure their accesses to disks, they need to implement distributed access control and locking for the disks. This leads to the usage of the Paxos protocol and its variants, as is done, e.g., in Petal [18] and Frangipani [24], using disks as memory for information sharing and coordination. Moreover, it is attractive to implement a higher-level SMR based application in such an environment since replicated data persistence is achieved at no additional cost.

Client-Server Model : In client-server applications with a potentially very large and dynamic set of clients, a highly available service can be implemented by a replicated set of servers distributed over a

*This work was supported in part by the Israeli Ministry of Science grant #1230-3-01.

wide-area network. In such a setting, restricting the servers' role to that of a persistent shared memory and letting clients carry out distributed algorithms has the following advantages:

- The complexity of failure monitoring and reconfiguration which is manifested, e.g., in group communication middlewares [21] is avoided.
- When there is risk of malicious penetrations, the damage caused to the system by malicious clients is much lower than that caused by malicious servers. This is because the servers hold replicas of many different objects, whereas client accesses are limited to their "allowed" objects. Thus, by means of per-object access control, it is possible to make it infeasible for a potential attacker to corrupt the objects of other clients. Consequently, in such systems it would be reasonable to assume a benign failure model for clients and a Byzantine failure model for the replica servers. Thus, the advantage of the shared memory model in such systems becomes apparent: By letting clients play the active role in the protocol for ordering operations, we alleviate the need for costly cryptographic protocols and require only simple access control mechanism. This approach was manifested in the Fleet survivable object repository [19].
- Moving the protocol complexity to clients reduces the server load. This is particularly true for systems where the number of servers is small relative to the number of clients. Since today's client applications are typically run by powerful desktop computers, this transition seems to be rather natural.
- In today's business applications, data is frequently stored on dedicated database servers (such as SQL servers), or on legacy systems such as mainframes. The functionality of such systems is typically limited to that of database query processing - it is considered infeasible to run application protocols on such systems. The way in which the industry deals with this issue is by adding another middle-tier application server layer whose responsibility is to handle high-level application protocols. Our approach allows us to eliminate the middle-tier altogether, in order to save on communication and prevent the middle-tier server from becoming a bottleneck. Alternatively, it allows us to create a dynamic and flexible application server that is composed of a dynamic group of processes, each of which runs as a Paxos client. The advantage in using the fixed databases to share information is that the middle-tier processes will not need to communicate with one another, nor to monitor changes and respond in any way to a reconfiguration of the tier (though clients of the application service will obviously need to be informed somehow of how to access the service, and of changes in it).
- From the perspective of programming convenience, shared memory algorithms are usually much simpler and more concise than their message-passing counter-parts. In addition, distributed protocols which preserve the client-server structure are more suitable for integration into existing client-server systems than peer-group protocols.

Indeed, supporting SMR in the shared memory model was recently proposed by Gafni and Lamport in [10] for one of the environments listed above, a SAN. Their protocol is run by clients that use network-attached commodity disks as read/write shared memory. The protocol assumes that up to a minority of the disks can fail by crashing. In Disk Paxos, each disk stores an array with an entry for each participating client. Each client can read the entries of all the clients but can write only its own entry. Each of the two Paxos phases is simulated by writing a ballot to the process entry at a majority of disks, and then reading other process entries from a majority of disks to determine whether the ballot has succeeded.

A fundamental limitation of Disk Paxos, which is inherited from all known variants of the Paxos protocol, is its inherent dependence on a priori knowledge of the number and the identities of all potential clients. The consequences of this limitation are twofold: First, it makes the protocol inappropriate for deployment in dynamic environments, where network disks are accessed from both static desktop computers and mobile devices (e.g., PDAs and notebooks computers). Second, even in stationary clusters, it poses scalability and management problems since in order for new clients to gain access to the disks, they should either forward their requests to a dedicated server machine, or first undergo a costly join protocol which involves real-time locking [10].

In this position paper, we offer to tackle coordination of infinitely many clients using the Paxos approach. In [6], we initiated a study of the Paxos algorithm in a very general shared memory model, in which both processes and memory registers can be faulty, the number of clients that can access the memory is unlimited, and the client identities are not known. In the next section we summarize our results and outline future directions.

2 Paxos with faulty shared memory and infinitely many processes

It is first worth noting how solving Consensus in the asynchronous shared memory model with infinitely many processes differs in a formal way from the finite case. To see this, consider the *failure-free termination* condition, which only requires processes to decide in executions where there are no process failures. Then it is possible to solve Consensus with failure-free termination in the asynchronous shared memory model using only read/write registers for any finite number of processes. However, it is impossible to solve Consensus with failure-free termination for infinitely many processes using only finite read/write memory. This result can be obtained using a technique similar to that of [3].

In practice, processes may fail, adding more complexity. The usual approach to achieve fault tolerance is to augment the system with an unreliable leader election oracle Ω [5]. The leader election service does not need to be always safe, and may allow multiple leaders to exist at times. However, in order to guarantee progress, it must eventually and for a sufficiently long time provide an exclusive leader. Equipped with Ω, Consensus is possible for finitely many processes using read/write registers, but even with Ω, a finite number of read/write registers is not sufficient to implement Consensus among infinitely many clients.

Furthermore, the presence of infinitely many clients implies that the specification of Ω itself should be modified. Intuitively, a desirable specification should be powerful enough to solve Consensus, and in the same time be implementable under some reasonable system assumptions (such as e.g., partial synchrony). But even during the system stability periods, it is unrealistic to require the failure detector to output an exclusive leader forever, unless some bounds are assumed on the maximum number of clients that can potentially or concurrently contend for becoming a leader. For example, in [7] we have shown an explicit construction of a probabilistic mutual exclusion primitive which guarantees that eventually a single client is granted access to the critical section (with probability 1) if the number of concurrently contending clients is bounded (but unknown). Other examples of such restricting assumptions can be found in [20].

Our solution first breaks the Paxos protocol using an abstraction of a shared object called a *ranked register*, which follows a recent deconstruction of Paxos by Boichat et al. in [2][1]. Briefly, a ranked register supports rr-*read* and rr-*write* operations that are both parameterized by an integer (the rank/ballot). The main property of this object is that a rr-*read* with rank r_1 is guaranteed to "see" any completed rr-*write* whose rank r_2 satisfies $r_1 > r_2$, i.e., the rr-*read* returns the value written in the rr-*write* or a later one. In order for this property to be satisfied, some lower ranked rr-*write* operations that are invoked after a rr-*read* has returned must *abort*. Armed with this abstract shared object, we provide a simple implementation of Paxos-like agreement using one reliable shared ranked register that supports infinitely many clients.

The remarkable feature of the ranked register is that while being strong enough to solve Consensus among an unlimited number of clients when Ω is present, it is nevertheless weak enough to be implementable in our system. Moreover, the Herlihy Consensus number [12] of the ranked register is only 1, the same as read/write registers, and it can be implemented for finitely many processes using a finite number of read/write registers (see [10]). A ranked register is implementable in the shared memory model with non-responsive crash failures. In [6] we present a wait-free construction of a ranked register out of n single ranked registers, of which $\lfloor (n-1)/2 \rfloor$ can incur non-responsive crash failure. In [6] we also prove that it is impossible to implement the ranked register with failure-free termination for infinitely many processes using only a finite number of read/write registers.

Due to its simplicity, a single ranked register can be easily implemented in hardware with the current Application Specific Integrated Circuit (ASIC) technology. Thus, the immediate application of the ranked register would be an improved version of Disk Paxos that supports unmediated concurrent data access by an unlimited number of processes. This would only require augmenting the disk hardware with the ranked

[1]In [2], an abstraction called *round-based register* is introduced, which we use but modify its specification.

register which would be readily available in Active Disks [22, 1] and in Object Storage Device controllers [11].

As part of our future work, we intend to conduct a comprehensive study of Paxos in the shared memory model with faults and unlimited number of clients. The issues of particular interest include specifying a leader election oracle and treatment of arbitrary non-responsive memory failures. Another promising direction would be to use the ranked register abstraction as a machinery for mapping numerous Paxos implementations found in the literature to the shared memory framework. This work will provide us with important insights of how various optimization techniques proposed in the context of other Paxos implementations can be applied to the shared memory counterpart. It is also interesting to explore the Paxos ability to solve Consensus among infinitely many processes under various assumptions on concurrency and process participation as defined in [20].

Acknowledgments

We are thankful to Ittai Abraham for helpful discussions.

References

[1] A. Acharya M. Uysal and J. Saltz. Active Disks: Programming model, algorithms and evaluation. In Proceedings of *the 8th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VIII)*, October 1998.

[2] R. Boichat, P. Dutta, S. Frolund and R. Guerraoui. Deconstructing Paxos. *Technical Report DSC ID:200106*, Communication Systems Department (DSC), École Polytechnic Fédérale de Lausanne (EPFL), January 2001.
Available at `http://dscwww.epfl.ch/EN/publications/documents/tr01_006.pdf`.

[3] J. Burns and N. Lynch. Bounds on shared memory for mutual exclusion. *Information and Computation* 107(2):171–184, December 1993.

[4] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation*, February 1999.

[5] T. D. Chandra, V. Hadzilacos and S. Toueg. The weakest failure detector for solving consensus. *Journal of the ACM* 43(4):685–722, July 1996.

[6] G. Chockler and D. Malkhi. Active disk Paxos with infinitely many processes. In *Proceedings of the 21st ACM Symposium on Principles of Distributed Computing (PODC'02)*, July 2002. To appear.

[7] G. Chockler, D. Malkhi and M. K. Reiter. Backoff protocols for distributed mutual exclusion and ordering. In Proceedings of the *21st International Conference on Distributed Computing Systems*, pages 11-20, April 2001.

[8] R. DePrisco, B. Lampson and N. Lynch. Fundamental study: Revisiting the Paxos algorithm. *Theoretical Computer Science* 243:35–91, 2000.

[9] 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, April 1985.

[10] E. Gafni and L. Lamport. Disk Paxos. In Proceedings of *14th International Symposium on Distributed Computing (DISC'2000)*, pages 330–344, October 2000.

[11] G. A. Gibson, D. F. Nagle, K. Amiri, J. Butler, F. W. Chang, H. Gobioff, C. Hardin, E. Riedel, D. Rochberg and J. Zelenka. A cost-effective high-bandwidth storage architecture. In Proceedings of the *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, October 1998.

[12] M. Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems* 11(1):124–149, January 1991.

[13] I. Keidar and D. Dolev. Totally ordered broadcast in the face of network partitions: exploiting group communication for replication in partitionable networks. In *Dependable Network Computing, Chapter 3*, D. Avresky Editor, Kluwer Academic Publications. January 2000.

[14] L. Lamport. Time, clocks, and the ordering of events in distributed systems. *Communications of the ACM* 21(7):558–565, July 1978.

[15] L. Lamport. The Part-time parliament. *ACM Transactions on Computer Systems* 16(2):133–169, May 1998.

[16] L. Lamport. Paxos made simple. *Distributed Computing Column of SIGACT News* 32(4):34–58, December 2001.

[17] B. W. Lampson. How to build a highly available system using Consensus. In Proceedings of the *10th International Workshop on Distributed Algorithms (WDAG)*, Springer-Verlag LNCS 1151:1-17, Berlin, 1996,

[18] E. K. Lee and C. Thekkath. Petal: Distributed virtual disks. In Proceedings of the *7th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-VII)*, pages 84-92, October 1996.

[19] D. Malkhi and M. K. Reiter. An architecture for survivable coordination in large-scale systems. *IEEE Transactions on Knowledge and Data Engineering* 12(2):187–202, March/April 2000.

[20] M. Merritt and G. Taubenfeld. Computing with infinitely many processes. In Proceedings of *14th International Symposium on Distributed Computing (DISC'2000)*, pages 164–178, October 2000.

[21] D. Powell, editor. Group communication. *Communications of the ACM* 39(4), April 1996.

[22] E. Riedel, C. Faloutsos, G. A. Gibson and D. Nagle. Active Disks for large-scale data processing. *IEEE Computer*, June 2001.

[23] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys* 22(4):299–319, December 1990.

[24] C. Thekkath, T. Mann and E. K. Lee. Frangipani: A scalable distributed file system. In Proceedings of the *16th ACM Symposium on Operating Systems Principles*, pages 224–237, October 1997.