A Data-Centric Approach for Scalable State Machine Replication [⋆]

Gregory Chockler[1], Dahlia Malkhi[1], and Danny Dolev[1]

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

1 Introduction

Data replication is a key design principle for achieving reliability, high-availability, survivability and load balancing in distributed computing systems. The common denominator of all existing replication systems is the need to keep replicas consistent. The main paradigm for supporting replicated data is *active replication*, in which replicas execute the same sequence of methods on the object in order to remain consistent. This paradigm led to the definition of *State Machine Replication* (SMR) [9, 14]. The necessary building block of SMR is an engine that delivers operations at each site in the same total order without gaps, thus keeping the replica states consistent.

Traditionally, existing SMR implementations follow a *process-centric* approach in which processes actively participate in active replication protocols. These implementations are typically structured as a peer group of server processes that employ group communication services for reliable totally ordered multicast and group membership maintenance. The main advantage of this approach is that during stability periods, work within a group is highly efficient. However, when failures occur and are detected the system needs to reconfigure. This requires solving agreement on group membership changes. Moreover, membership maintenance implies that participants need to constantly monitor each other, yielding n^2 probe complexity. Consequently, group communication based systems scale poorly as the group size and/or its geographical span increases. Additionally, due to the high cost of configuration changes, these solutions are not suitable for highly dynamic environments.

In contrast, we advocate the use of a *data-centric* replication paradigm in order to alleviate the scalability problems of the process-centric approach. The main idea underlying the *data-centric* paradigm is the separation of the replication control and the replica's state. This separation is enforced through a two-tier architecture consisting of a *storage tier* whose responsibility is to provide persistent storage services for the object replicas, and a *client tier* whose responsibility is to carry out the replication support protocols. The storage tier is comprised of logical storage elements which in practice can range from network-attached disks to full-scale servers. The client tier utilizes the storage tier for communication and data sharing thus effectively emulating a shared memory environment.

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

The data-centric approach promotes fundamentally different replication solutions. First, it fits today's state-of-the-art Storage Area Network (SAN) environments, where disks are directly attached to high speed networks that are accessible to clients. Also, there is no need for proprietary communication layers and/or tools (such as group communication). In fact, all the communication can be carried out over a standard RPC-based middleware such as Java RMI or CORBA. Third, storage elements need not communicate with one another or monitor each other, nor is there need for reconfiguration upon failures. This reduces the cost of fault-management and enhances scalability. Replication groups can be created on-the-fly by clients simply by writing the initial object state and code to storage elements of their choice. Fault-tolerance can be achieved by means of quorum replication as it is done in the Fleet survivable object repository [11].

The data-centric approach is capable of supporting replication in highly dynamic environments where the replicas are accessed by an unbounded number of possibly faulty clients whose identities are not known in advance. In this paper, we show how to realize this by extending the well-known Paxos approach of Lamport [10] to a very general shared memory model, in which both processes and memory objects can be faulty, the number of clients that can access the memory is unlimited, and the client identities are not known.

2 SMR in Data Centric Environments

The Paxos algorithm of Lamport [10] is one of the most widely used techniques for implementing operation ordering for SMR. Numerous flavors of Paxos that adapt it for various settings and environments have been described in the literature. At the core of Paxos is a Consensus algorithm called *Synod.* Since Consensus is unsolvable in asynchronous systems with failures [5], the Synod protocol, while guaranteeing always to be safe, ensures progress 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 Paxos implementations were designed for process-centric environments, where the replicas in addition to being data holders, also actively participate in the ordering protocol. Recently, Gafni and Lamport proposed a protocol for supporting SMR in the shared memory model [6] emulated by the SAN environment. 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 that involves real-time locking [6].

In [3], 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. Our results are summarized in the following section.

2.1 Paxos with faulty shared memory and infinitely many processes

Solving Consensus in an asynchronous shared memory model with infinitely many processes differs in a formal way from the finite case. To see this, consider a *failure-free termination* condition, which only requires processes to decide in executions where there are no process failures. 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 easy to show impossibility of Consensus with failure-free termination for infinitely many processes using only finite read/write memory.

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 [2]. 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 (again) 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 at the same time be implementable under some reasonable system assumptions (such as 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 [4] we have shown an explicit construction of a probabilistic mutual exclusion primitive that 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 [12].

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 [1][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, any lower ranked rr-*write* operation that is 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 [8] 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 [6]). A ranked register is implementable in the shared memory model with non-responsive crash failures. In [3] 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 [3] 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 register, which would be readily available in Active Disks and in Object Storage Device controllers (see, e.g., [7]).

3 Future Directions

To better understand the power of our approach, a comprehensive study of Paxos in the shared memory model with faults and unlimited number of clients need to be conducted. The issues of particular interest include specifying a leader election oracle and treatment of non-responsive arbitrary memory failures, which in practice correspond to replicas exhibiting a malicious behavior.

Interestingly, tolerating malicious memory failures appears to have several non-obvious consequences with respect to the overall number of shared memory objects needed to achieve the desired resilience level. In particular, in contrast to the well-known $3f+1$ lower bound on the number of processes needed to tolerate up to f Byzantine failures in the message passing model, all the existing wait-free algorithms for asynchronous shared memory model with faults require at least $4f+1$ memory objects to tolerate malicious failures of at most f memory objects (see e.g., [11]). In this respect, it seems an important future direction

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

to study the computational power of this model in order to establish whether the more pessimistic resilience guarantees are needed to overcome the inherent model limitations, or just have to deal with the algorithm efficiency.

Another interesting direction would be to use the ranked register abstraction as a machinery for unifying numerous Consensus implementations found in the literature. Of particular interest is the class of so called *indulgent* Consensus algorithms: i.e., the algorithms designed for asynchronous environments augmented with an unreliable failure-detector. The Synod algorithm of Paxos is an example of such indulgent algorithm. Another example is the revolving-coordinator protocol (e.g., see [2]), which is based on a similar principle as Paxos but has the leader election being explicitly coded into the algorithm. One of the benefits of establishing a uniform framework for asynchronous Consensus algorithms will be in a better understanding of how the lower bounds for Consensus in asynchronous message passing model can be matched in its shared memory counterpart.

On a more practical note, we envision that in the years to come the worlds of distributed computing and Storage Area Networks will continue to converge. It is already apparent that many problems arising in the design of practical SAN based software systems closely resemble those found in such extensively studied areas of distributed computing as concurrency control, mutual exclusion, agreement, etc. However, there is still a gap between the distributed computing theory and practical SAN systems. Bridging this gap will be a striking experience for both SAN system architects and the distributed computing research community.

References

1. 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`.
2. T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM* 43(2):225–267, March 1996.
3. 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.
4. 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.
5. 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.
6. E. Gafni and L. Lamport. Disk Paxos. In Proceedings of *14th International Symposium on Distributed Computing (DISC'2000)*, pages 330–344, October 2000.
7. 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.
8. M. Herlihy. Wait-free synchronization. *ACM Transactions on Programming Languages and Systems* 11(1):124–149, January 1991.

9. L. Lamport. Time, clocks, and the ordering of events in distributed systems. *Communications of the ACM* 21(7):558–565, July 1978.
10. L. Lamport. The Part-time parliament. *ACM Transactions on Computer Systems* 16(2):133–169, May 1998.
11. 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.
12. 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.
13. D. Powell, editor. Group communication. *Communications of the ACM* 39(4), April 1996.
14. F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys* 22(4):299–319, December 1990.