1. A Data-Centric Approach for Scalable State Machine Replication *

Gregory Chockler[1,2], Dahlia Malkhi[1], and Danny Dolev[1]

[1] School of Computer Science and Engineering, The Hebrew University of Jerusalem, Jerusalem, Israel 91904
email:{grishac,dalia,dolev}@cs.huji.ac.il
[2] IBM Haifa Labs, Tel-Aviv Group

1.1 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) [1.8, 1.13]. 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. 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

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

utilizes the storage tier for communication and data sharing thus effectively emulating a shared memory environment.

The data-centric approach promotes fundamentally different replication solutions: First, it perfectly fits today's state-of-the-art Storage Area Network (SAN) environments, where disks are directly attached to high speed networks, usually Fibre Channel, that are accessible to clients. Moreover, due to the lack of broadcast support by Fibre Channel networks, direct multicast communication among processes, as mandated by the process-centric paradigm, is not easily supportable in SAN environments. Second, it simplifies the system deployment and management as the only deployment prerequisite is the existence of an infrastructure of storage elements which can be completely dynamic. Also, there is no need to deploy any non-standard 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, the storage elements do not need to communicate with one another nor they need to monitor each other and reconfigure upon failures. This reduces the cost of fault-management and increases the system scalability. The replication groups can be created on-the-fly by clients simply writing the initial object state and code to some pre-defined collection of the storage elements. Fault-tolerance can be achieved by means of quorum replication as it is done in the Fleet survivable object repository [1.11].

Finally, the data-centric approach has a potential for supporting replication in highly dynamic environments where the replicas are accessed by an unlimited number of possibly faulty clients whose identities are not known in advance. In this paper, we first mention the results introduced in [1.3], which extend the Paxos approach of Lamport [1.9] to such environments. These results provide universal object emulation in 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. We then outline future directions for research within the data-centric framework.

1.2 SMR in Data Centric Environments

The Paxos algorithm of Lamport [1.9] is one of the techniques used to implement 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 [1.5], 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 [1.6] 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 [1.6].

In [1.3], 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. Since wait-free Consensus is impossible even for two processes in this model [1.7], we augment the system with an unreliable leader oracle, which guarantees that eventually and for a sufficiently long time some process becomes an exclusive leader. Equipped with a leader oracle, Consensus is possible for finitely many processes using read/write registers. However, even with a leader oracle, an infinite number of read/write registers is necessary to implement Consensus among infinitely many clients (see [1.3] for the proof).

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][1]. The remarkable feature of the ranked register is that while being strong enough to guarantee Consensus safety regardless of the number of participating clients, it is nevertheless weak enough to allow implementations satisfying wait-free termination in runs where any number of clients might fail. Armed with this abstract shared object, we provide

[1] In [1.1], an abstraction called *round-based register* is introduced, which we use but modify its specification.

a simple implementation of Paxos-like agreement using one reliable shared ranked register that supports infinitely many clients.

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.

1.3 Future Directions

In contrast to the process-centric approach whose power and limitations are now well understood, the data-centric paradigm poses several unresolved theoretical questions requiring a further study. First, let us take a closer look on the oracle abstraction required to guarantee the liveness of the Consensus protocol. Most of the oracle abstractions found in the literature are targeted for message passing systems. An exception is found in [1.10] where the notion of an unreliable failure detector was adapted to the shared memory model (without memory faults). Still, this approach is not easily implementable with infinitely many processes, as it assumes that processes are able to monitor each other which is problematic if the process universe is a priori unknown.

On the other hand, weak synchronization paradigms match the shared memory environment more closely and as demonstrated in [1.4], can be made oblivious to the overall number of clients and their identities. In light of this, an interesting future direction is to identify a class of weak synchronization primitives sufficient for solving Consensus in an asynchronous system, and study their implementability in presence of infinitely many processes.

The next future direction is concerned with tolerating malicious memory failures. Interestingly, this 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., [1.11]). Martin et al. show in [1.12] an implementation of an atomic register with $3f+1$ servers which utilizes enhanced server functionality and increased communication. In particular, in their method clients *subscribe* to servers so as to repeatedly receive updates about the register's state. Further exploration is required to investigate this seeming tradeoff between the resilience on one hand, and the communication cost and memory consumption on the other hand.

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 [1.2]), 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.

Finally, on a more practical note, let us take a look on today's SAN based systems. These systems usually employ sophisticated software layers which emulate higher level abstractions such as virtual disks, object stores and file systems over the SAN. These software layers typically manage large volumes of metadata such as directory structure, access permissions, etc., whose availability and consistency are crucial. This mandates using replication to ensure continuous metadata availability despite faults. In this context, our data-centric SMR represents a scalable solution for maintaining metadata replicas in a consistent state. Consequently, integrating data-centric SMR into the SAN software is a challenging future direction of practical importance.

References

1.1 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`.

1.2 T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM* 43(2):225–267, March 1996.

1.3 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.

1.4 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.

1.5 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.

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

1.7 P. Jayanti, T. Chandra, and S. Toueg. Fault-tolerant wait-free shared objects. *Journal of the ACM* 45(3):451-500, May 1998.

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

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

1.10 W. K. Lo and V. Hadzilacos. Using failure detectors to solve consensus in asynchronous shared-memory systems. In Proceedings of the *8th International Workshop on Distributed Algorithms (WDAG)*, Springer-Verlag LNCS 857:280-295, Berlin, 1994.

1.11 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.

1.12 J. P. Martin, L. Alvisi and M. Dahlin. Minimal Byzantine Storage. In Proceedings of *the 16th International Conference on DIStribued Computing (DISC'02)*, pages 311–325, October 2002,

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