Optimal Resilience Wait-Free Storage from Byzantine Components: Inherent Costs and Solutions

Gregory Chockler*, Idit Keidar†, and Dahlia Malkhi‡

May 19, 2004

Abstract

This position paper describes the results of our on-going investigation into the possibility and cost of building a survivable store. It considers optimal resilience systems comprising of $3t+1$ base storage units, t of which may fail by becoming non-responsive or arbitrarily corrupted. Our contribution includes both algorithms and lower bounds in this model. We illuminate an inherent difficulty of achieving optimal resilience in the form of two lower bounds, on read and on write complexities. We also provide the first optimal-resilience wait-free algorithms that match these bounds in performance. Finally, we suggest some directions for future research.

Introduction. Replicating storage is a fundamental mechanism for fault tolerance. When storage can exhibit arbitrary corruption, replication can mask corrupted elements and guarantee survivability. The formal model capturing this setting is an asynchronous system with multiple processes accessing fault-prone shared memory objects [3, 2, 14]. We assume that a threshold t of the memory objects may fail by being non-responsive or by returning arbitrary values (i.e., by being Byzantine); this failure model was named *non-responsive arbitrary (NR-Arbitrary) faults* by Jayanti et al. [14]. An unbounded number of processes may access the shared memory objects, and these processes may fail by crashing. We focus on *wait-free* reliable storage solutions; that is, solutions that guarantee that all operations submitted by correct processes terminate despite the failure of an unbounded number of processes.

Over the past couple of years, our research in this area has focused on studying the limits of survivable storage in terms of a resilience threshold and protocol complexity. We contribute a full picture concerning the possibility and complexity of emulating survivable storage in our model out of $n = 3t + 1$ memory elements, including algorithms and lower bounds. Our work shows that optimal resilience algorithms have an inherent cost: We prove in [8] a lower bound of two rounds for emulating WRITE operations with a resilience of $t \geq n/4$. This is in contrast to algorithms tolerating $t < n/4$ NR-Arbitrary faults, which can emulate WRITE operations in a single round. Moreover, we show a lower bound of $min(t + 1, f + 2)$ rounds for emulating READ operations in runs with f failures in systems where the reader does not modify the base objects.

These bounds are tight: In an accompanying paper [1], we provide the first emulation of a wait-free safe regular register out of $3t + 1$ registers, whose read and write complexities are optimal according to the above lower bounds. That work also presents a full Byzantine shared-memory Paxos algorithm built with the reliable register emulation.

Motivation. A "storage centric" approach for service replication, discussed by Malkhi in [20], models the system as a fault-prone shared memory model. This paradigm captures a fair amount of recent work, that comes in three main flavors:

1. One flavor is message-passing client-server systems in which servers store information on behalf of clients and the only communication is between clients and servers. Scalability is achieved by making

*CS and AI Lab, MIT.

†Department of Electrical Engineering, The Technion – Israel Institute of Technology.

‡School of Computer Science and Engineering, The Hebrew University of Jerusalem.

servers as light as possible. Examples of systems built using this approach include Fleet [21], SBQ-L [22], Agile Store [15], Coca [25], and [5].

2. The second flavor is given by today's peer-to-peer systems. These systems consist of a collection of nodes spread all over the Internet that store data objects. The storage nodes are accessed by a potentially large, dynamically changing client universe. Naturally, due to their Internet-wide deployment, the storage nodes are prone to malicious attacks. Also, a storage node may simply drop out of the system and be replaced by another node that has no knowledge of updates received by the original node. This motivates adopting a Byzantine failure model for the storage nodes. Examples of peer-to-peer systems that adopt storage-centric replication to support availability in face of Byzantine failures include Rosebud [23] and [18].

3. The third flavor directly expresses an emerging network technology, the Storage Area Network (SAN). 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. Examples of SAN-based systems that use disks for information sharing and coordination include Compaq's Petal [17] and Frangipani [24], Disk Paxos [13], and Active Disk Paxos [9].

Our goal is to enhance this fruitful line of work and to investigate the costs and tradeoffs involved in designing reliable storage solutions for scalable decentralized systems that may be deployed in one of the two settings described above. Our focus is on a survivable distributed storage system that tolerates arbitrary corruption and unresponsiveness (i.e., NR-Arbitrary faults) in up to a third of its disks (or servers) as well as process crashes. (Tolerating NR-Arbitrary faults in a third or more of the disks is impossible [22]).

Previous techniques. Previous work that constructed wait-free objects in this failure model used $4t+1$ [21], $5t+1$ [14], or even $6t+1$ [11] base objects. A natural question to ask is whether the resilience threshold $t < n/4$ is tight in this model. Intuitively, one would hope for a resilience bound of $t < n/3$. Several previous works have addressed this question, and have achieved better resilience by weakening the model in different ways – by adding synchrony [5]; by storing authenticated (signed) self-verifying data at the servers [21, 22]; by hinging on active storage servers [22], or by providing solutions that may block indefinitely if clients do fail [22, 4]. However, $t < n/4$ is the best resilience threshold previously achieved for wait-free constructions in the model considered herein.

In contrast, the literature is abundant with message-passing consensus algorithms that are resilient to Byzantine failures of less than a third of the processes. Therefore, an appealing way to go about searching for a more resilient solution would be to try and adapt the techniques used in those algorithms to our model. Two basic techniques are used in achieving this threshold in consensus algorithms: authentication (using signatures) of all transmitted data (e.g., [12, 7]) or echoing (e.g., [6, 12]). Authentication can indeed be used to construct memory objects that are resilient to $t < n/3$ failures (see [21]). However, our model does not incorporate a signature scheme as needed for authentication. The second approach is to have each process echo all the values it receives to all the processes (e.g., see [12]). Unfortunately, echoing cannot help us address the challenge we have set out to solve. Indeed, if a correct process can correctly echo information to all other processes, this is essentially the same as having a wait-free register through which the process conveys the information to the other processes. And implementing such a register from fault-prone storage is exactly what we seek to do.

Lower bounds. Having ruled out the use of standard techniques to improve the resilience threshold, we proceeded to examine whether there are any inherent limitation that prevent algorithms in our model from achieving better resilience. We observe that existing algorithms for fault-prone shared memory models (e.g., [21, 22, 5, 14, 4]) implement (emulate) READ and WRITE operations in a single round; that is, they invoke one read or write operation on each base object. Our first lower bound in [8] proves that if $t \geq n/4$, then it is impossible to emulate the WRITE operations of a wait-free register by invoking a single round of operations on the base objects. Our proof applies to binary single-writer single-reader *safe registers*, the weakest meaningful register type [16].

We further show that if $n = 3t + 1$, then any algorithm in which the reader does not modify the base objects' states may need to invoke as many as $t + 1$ rounds of read operations on base objects in order to emulate a single READ operation of a single-writer single-reader safe register. More generally, for any $0 \leq f \leq t$, there is a run in which f objects are Byzantine faulty in which the algorithm invokes $min(t + 1, f + 2)$ rounds of base object operations.

Constructions. Our lower bounds are matched by an algorithm in an accompanying paper [1], thus providing a complete picture of shared-memory survivability with $3t + 1$ registers. A single-writer multi-reader safe register is the most basic building block that can be used for constructing various object types. E.g., there are well-known constructions of regular and atomic registers from safe ones [16]. These, in turn, can be used to construct higher level objects; e.g., consensus can be solved with regular registers if the system is augmented with an oracle failure detector [19].

Discussion. We now explain how our lower bounds apply to the two existing optimal resilience register constructions of [22, 4]. First, in both [22] and [4], the writer invokes only a single round of object operations (assuming a single writer). This raises the question of how the necessity of two object invocation rounds is circumvented. The reason for this is that the emulation algorithms described in these two papers are not required to satisfy wait-freedom. In particular, the reader might never return in runs where the writer fails in the middle of writing a value but the reader is correct and invokes READ after the writer fails.

In order to circumvent the read lower bound, the algorithm of [22] assumes a push-like communication model for the READ implementation, where the reader subscribes to the object updates when it first contacts a base object. Subsequently, each correct base object keeps informing each of its subscribers of all the modifications it undergoes. In contrast, our communication model (for both read and write) is the standard shared memory request-response model (i.e., pull-like), and therefore, the reader is forced to continuously invoke additional read rounds in order to discover these modifications. Finally, the number of rounds invoked by the READ emulation of [4] may be unbounded, and therefore, matches our lower bound of $min(t+1, f+2)$ rounds. This is not surprising since the READ implementation of [4] is not allowed to modify the base object states and therefore, (as our lower bounds imply) will not be able to terminate in less than $min(t+1, f+2)$ rounds in some runs.

Future directions. One of the research goals we will pursue in the near future is to design a survivable storage system, in which the user has complete autonomy in defining the replication group for each file or data object, and in which all protocols are confined to the objects they operate on. Toward that end, we have recently devised light-weight storage-centric *leases* in [10], and an effort is under way to conduct performance measurements of leases in real-life LANs.

In the process of designing survivable storage solutions, we will strive to identify good building blocks (abstractions) of which such systems can be composed. For example, the results summarized in this paper indicate that wait-free registers have an inherent cost. An interesting future direction is to search for weaker basic abstractions that would be cheaper and simpler to implement, and yet would serve in the construction of survivable storage solutions. Of course, in order to realize this research direction, one must try to build a reliable store with the desirable semantics out of the suggested abstractions.

References

[1] I. Abraham, G. Chockler, I. Keidar, and D. Malkhi. Byzantine Disk Paxos. Submitted for publication, 2004.

[2] Y. Afek, D. Greenberg, M. Merritt, and G. Taubenfeld. Computing with faulty shared objects. *Journal of the ACM*, 42(6):1231–1274, November 1995.

[3] Y. Afek, M. Merritt, and G. Taubenfeld. Benign failures models for shared memory. In *Proceedings of the 7th International Workshop on Distributed Algorithms*, pages 69–83. Springer Verlag, September 1993. In: *LNCS 725*.

[4] H. Attiya and A. Bar-Or. Sharing memory with semi-byzantine clients and faulty storage servers. In *SRDS*, 2003.

[5] R. Bazzi. Synchronous byzantine quorum systems. *Distributed Computing*, 13(1):45–52, 2000.

[6] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM*, 32(4):824–840, October 1985.

[7] M. Castro and B. Liskov. Practical byzantine fault tolerance. In *The 3rd Symposium on Operating Systems Design and Implementation (OSDI '99)*, February 1999.

[8] G. Chockler, I. Keidar, and D. Malkhi. The inherent cost of optimal resilience wait-free storage from byzantine components. Submitted for publication, 2004.

[9] 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)*, 2002.

[10] G. Chockler and D. Malkhi. Light-weight leases for storage-centric coordination. Technical Report MIT-LCS-TR-934, MIT Laboratory for Computer Science, 2004.

[11] 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, 2001.

[12] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, Apr. 1988.

[13] E. Gafni and L. Lamport. Disk paxos. *Distributed Computing*, 16(1):1–20, 2003.

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

[15] S. Lakshmanan, M. Ahamad, and H. Venkateswaran. Responsive security for stored data. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*, 2003.

[16] L. Lamport. On interprocess communication – part ii: Algorithms. *Distributed Computing*, 1(2):86–101, 1986.

[17] 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, 1996.

[18] S. Lin, M. C. Q. Lian, and Z. Zhang. A practical distributed mutual exclusion protocol in dynamic peer-to-peer systems. In *3rd International Workshop on Peer-to-Peer Systems (IPTPS'04)*, 2004.

[19] 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)*, number 857, pages 280–295. Springer-Verlag LNCS, 1994.

[20] D. Malkhi. From byzantine agreement to practical survivability; a position paper. In *Proceedings of the International Workshop on Self-Repairing and Self-Configurable Distributed Systems (RCDS 2002)*, 2002.

[21] D. Malkhi and M. Reiter. An architecture for survivable coordination in large distributed systems. *IEEE Transactions on Knowledge and Data Engineering*, 12(2):187–202, 2000.

[22] J.-P. Martin, L. Alvisi, and M. Dahlin. Minimal byzantine storage. In *Proceedings of the 16th International Symposium on Distributed Computing (DISC 2002)*, October 2002.

[23] R. Rodrigues and B. Liskov. Rosebud: A Scalable Byzantine-Fault-Tolerant Storage Architecture. Technical Report MIT-LCS-TR-932, MIT Laboratory for Computer Science, 2004.

[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, 1997.

[25] L. Zhou, F. B. Schneider, and R. van Renesse. Coca: A secure distributed on-line certification authority. *ACM Transactions on Computer Systems*, 20(4):329–368, 2002.