Position Paper: BFT: the Time is Now

Allen Clement[1], Mirco Marchetti[2], Edmund Wong[1],
Lorenzo Alvisi[1], and Mike Dahlin[1]
[1]The University of Texas at Austin, [2],University of Modena and Reggio Emilia,
{aclement, mirco, elwong, lorenzo, dahlin}@cs.utexas.edu

1 Introduction

Data centers strive to provide reliable access to the data and services that they host. This reliable access requires the hosted data and services hosted by the data center to be both *consistent* and *available*. Byzantine fault tolerance (BFT) replication offers the promise of services that are consistent and available despite arbitrary failures by a bounded number of servers and an unbounded number of clients.

The thesis of this position paper is simple: BFT is on the verge of becoming a practical reality—but clearing the last hurdles will require to rethink, once again, how BFT systems must be designed and implemented.

Three fundamental trends support our thesis that widespread adoption of Byzantine fault tolerance is at hand.

First, falling hardware costs and the increased value and importance of services are making significant non-BFT replication a standard commercial practice [5, 12, 13]. Although fault tolerance has long been an after-thought for non-critical applications, it is becoming increasingly worthwhile to use hardware generously to defend against component failures and geographic catastrophes [20]. For example, the Google file system (GFS) relies on three-way replication as a way to protect data from crash failures [13].

Second, in systems where both reliability and availability are important properties, crash tolerance is not enough. Byzantine faults are a frequent occurrence in the wild, manifesting themselves as disks that do not operate in a fail-stop manner [4], file systems that implement inadequate actions to recover from disk faults [22], file systems with bugs in their crash recovery code [25, 26], human errors [14, 21], or processors that exhibit transient Byzantine behavior because of soft errors [19, 23] and so on. Simple hardware failures can have dramatic consequences. For example, a single malfunctioning NIC at Los Angeles International Airport stalled immigration for more than 12 hours [8, 10].

Third, ten years of research in practical Byzantine fault tolerance [7, 6, 1, 11, 18, 17, 15, 16, 24, 27] have reduced significantly the costs traditionally associated with BFT. Throughput and latency of BFT replicated services are now competitive with those offered by their unreplicated counterparts [17] and, by separating agreement from execution [27], the effective overhead for BFT replication has been brought in line with the 3-way replication used by existing commercial systems such as GFS.

In summary, we believe that we are on the verge of reaching the inflection point where it becomes advantageous to trade increasingly inexpensive hardware for the piece of mind provided by BFT replication. However, we also believe that, unless the community changes fundamentally its approach towards building high performance BFT systems, this opportunity may be lost.

From PBFT [6, 7] to Zyzzyva [17], the design and implementation of high performance BFT systems has adopted the common-sense engineering principle of *optimize for the common case*, carefully tuning their systems to be extremely efficient during fault-free execution. We contend that this otherwise sound approach should be considered harmful when designing BFT system, as it encourages the proliferation of corner cases that result in dramatic performance degradations in the presence of faults. In other words, the design choices that have informed today's BFT systems may have made them too fragile for actual deployment: current BFT systems can safely *survive* Byzantine faults, but can hardly be said to *tolerate* them, as Byzantine faults can render these systems virtually unavailable.

We conclude by briefly reporting on Aardvark, our recent effort that demonstrates that true Byzantine fault tolerance can be achieved without sacrificing high performance. For complete details of the protocol and system design see [9].

Section 2 reviews the advances that have reduced the overheads of BFT replication. Section 3 exposes the dangers of over aggressive optimizations in existing BFT systems. Section 4 makes the case for a new approach in designing BFT systems which we demonstrate

2 Low Overhead BFT

Existing BFT replication systems achieve low overheads by paying careful attention to the *common case* in which executions are *gracious*. In PBFT [7] and Zyzzyva [17] a gracious execution is one in which the network is well behaved and there are no faulty clients or replicas; in Q/U [1] and HQ [11] a gracious execution has the additional constraint that client requests do not contend with each other. System designers are able to take advantage of properties of their chosen common case in order to provide excellent system throughput and reduce overheads as shown in the first column of Figure 1. In the remainder of this section we review the design of existing systems.

For review, Figure 2 illustrates the basic communication patterns in Castro and Liskov's PBFT protocol [7]. A set of $n \geq 3f+1$ servers select one of their number to be the *pri-*

System	Gracious Execution	Malformed MAC	Client Spam	Slow Primary			Replica Spam
				1ms	10ms	100ms	
PBFT [7]	36350	0	crash	5396	4635	1097	0
Zyzzyva [17]	48253	0	crash	14547	5141	crash	0
HQ[11]	15873	0*	0	N/A			0
Aardvark [9]	40527	40527	7873	38084	39089	37903	11706

Figure 1: Operations per second of BFT Replication protocols under various node behaviors when the network is well behaved and in the presence of 200 correct clients. Gracious execution corresponds to scenarios in which the network and all participating nodes are well behaved. The Malformed MAC column corresponds to scenarios in which a single client produces a MAC that can be authenticated by exactly one server. * The HQ prototype does not implement the full recovery path for malformed client MACs, so the attack cannot technically be implemented. The Client Spam column covers the case where a single faulty client floods the servers with 9000B messages. The Slow Primary column shows the throughput when a single primary introduces a delays of 1ms, 10ms, and 100ms before sending PRE-PREPARE messages. The Replica Spam columns show the impact of a replica that spams other servers with 9000B messages.

mary. A client sends its REQUEST message to the primary, and the primary assigns the request a sequence number and sends a PRE-PREPARE message to the other servers. The servers then do an all-to-all exchange of PREPARE messages and then of COMMIT messages. Once a sufficient number of servers agree on the request's order in a linearizable total order of all requests, they execute the request and send a REPLY message to the client. A clients acts on the REPLY once it has at least $f+1$ matching replies.

To ensure progress, a client retransmits a request to all replicas if it does not receive a reply by a timeout, and the replicas forward the request to the primary. Each replica then expects the request to complete execution of a request by a timeout. If no requests complete execution in time, the replica assumes that the primary is faulty and initiates a *view change* by stopping all processing of messages in the current view and sending a VIEW-CHANGE message to all servers. Once a sufficient number of replicas initiate a view change, they are able to start the next view with a different primary. Additionally, if a server falls behind in this asynchronous system, it is able to catch up by fetching a recent checkpoint and recent messages from its peers.

A key performance optimization is the use of message authentication codes (MACs) for authentication rather than digital signatures. In particular, REQUEST, PRE-PREPARE, PREPARE, and COMMIT messages contain an *authenticator*—an array of n MACs, one for each server. For practical values of n, generating n MACs is at least an order of magnitude faster than verifying a signature. For example, on a 2.0GHz Pentium-M, openssl 0.9.8g can compute over 500,000 MACs per second for 64 byte messages, but it can only verify 6455 1024-bit RSA signatures per second or produce 309 1024-bit RSA signatures per second.[1]

Other representative protocols are similar in principle, but vary their message patterns. For example, Zyzzyva [17] speculatively executes requests, replies to clients after the PRE-PREPARE phase, and can skip the subsequent steps if enough replies match. Q/U [1] eliminates the primary and uses client retransmissions to resolve conflicting updates. HQ [11] is a hybrid protocol that resembles Q/U in the absence of contention and relies on a protocol like PBFT rather than client back-off to resolve conflicts.

[1]Elliptic curve algorithms have faster signature generation (e.g., 2275 per second for 160-bit signatures, which are believed to be approximately equivalent in strength to 1024-bit RSA signatures) but slower signature verification (e.g., 499/s for 160-bit signatures); most PBFT messages are generated once and read $n-1$ times.

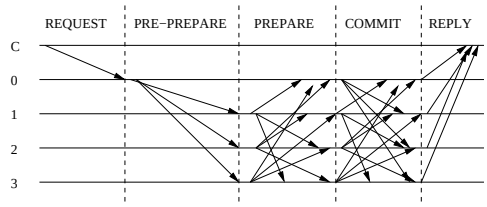

Figure 2: Basic communication pattern in PBFT

3 The Dark Secret of BFT

The dark secret of BFT replication protocols is that they rely heavily on gracious execution in order to maintain high throughput. One might hope that exotic "malicious server" attacks requiring careful coordination or an uncooperative network would be required to slow existing BFT systems dramatically; unfortunately existing protocol designs and implementations are sufficiently high strung that they can be disrupted by decidedly non-exotic behaviors including malformed messages from a single faulty client, primary, or replica even when the network is well behaved. Figure 1 shows the impact of a variety of Byzantine behaviors on the throughput of existing systems.

The malformed MAC column of Figure 1 shows the impact of a big MAC attack on system throughput. In a big MAC attack, a faulty client provides a MAC that can be authenticated by a single server but not by any other servers. When the primary orders the request, the other replicas are unable to authenticate its authenticity and a resolution sub protocol is required. This problem does not arise during gracious executions in which there are no faulty clients or servers and the network is well behaved. Unfortunately a single faulty

client can force the resolution protocol to be executed—additionally, a faulty primary or replica can request the resolution protocol and it is impossible for any node to ascertain whether the client, primary, replica, or network is at fault. Current implementations of PBFT [7] and Zyzzyva [17] observe a throughput of 0 when faced with this attack. HQ [11] is not vulnerable to the attack as described above since it does not rely on a primary, but a similar attack is possible. The attack cannot be implemented in HQ [11] since all clients and replicas are hard-coded to share the same keys.

The client and replica spam attacks in Figure 1 are implemented by a single process sending 9000 byte messages to all of the replicas from either a client or non-primary server machine. Current implementations of PBFT [7] and Zyzzyva [17] crash under the strain of these attacks and the throughput of HQ [11] is driven to 0 as the attacking node prevents legitimate clients from opening TCP connections to the servers.

The primary is a distinguished replica that is uniquely positioned to control the system's throughput—no request can be executed until it is assigned a sequence number by the primary. During gracious executions, the primary is always correct and provides optimal throughput. Previous authors [2, 3] have observed that a slow primary can have a substantial impact on system throughput. The slow primary column of Figure 1 shows the dramatic drop in system throughput when the primary delays ordering requests by 1, 10, and 100ms. HQ [11] does not rely on a primary to order requests so is not subject to a slow primary attack; the lack of a primary leaves HQ unable to batch requests and is the primary reason its peak throughput is a factor of 2 slower than PBFT and a factor of 3 slower than Zyzzyva.

Existing BFT systems guarantee consistency at all times and provide excellent performance during gracious executions in which there are no failures. Unfortunately, the design and implementation decisions made to achieve optimal performance during gracious executions leave the systems vulnerable to severe disruption by a single faulty client or server.

4 BFT: From Z to A

We argue that many BFT systems should be willing to give up some best case performance in order to provide good performance over a wider range of situations for two reasons.

First, in current systems the best case is fragile, so building a system around its best case performance may be dangerous. In particular, (1) the best case is achieved only when very strong assumptions hold and (2) departing from the best case can devastate performance because the system then provides at best *eventual progress* and at worst no practical progress. This fragility is not just a theoretical problem, as shown in Section 3.

Second, many systems may be insensitive to modest reductions in peak agreement throughput because of *limited demand* or *other bottlenecks.*

In particular, many services' peak demands are far under the best case throughput offered by existing BFT replication protocols. For such systems, *good enough is good enough*, and modest reductions in best case agreement throughput will have little effect on end to end system performance. In such systems, increased robustness may come at effectively no cost.

Similarly, when systems have other bottlenecks, Amdahl's law limits the impact of changing the performance of agreement. For example, Zyzzyva can execute about 50,000 null requests per second [17], suggesting that agreement consumes 20μs per request. If, rather than a null service, we replicate a service for which executing an average request consumes 100μs of processing time, then peak throughput with Zyzzyva would be about 8333 requests per second. If, instead, agreement were accomplished via a protocol with double the overhead of Zyzzyva (e.g., 40μs per request), peak throughput would still be about 7100 requests/second. In this hypothetical example, doubling agreement overhead reduces peak end-to-end throughput by less than 15%. Castro and Liskov [7] observed that the overheads of Byzantine consensus is in the noise compared to the overheads associated with running an NFS server.

In [9] we present Aardvark, a BFT replication system designed to provide good performance in all fault scenarios by avoiding common case optimizations that expose obscure corner cases. In the context of previous systems, Aardvark takes the surprising steps of judiciously relying on signatures during normal operation, frequently undergoing view changes in order to elect a new primary, and forgoing IP multicast for inter-replica communication. As shown in final row of Figure 1 these design decisions impose a 30% hit to throughput when compared to Zyzzyva, but result in a protocol that performs significantly better in the presence of failures.

5 Conclusion

Barbara Liskov observed that researchers spent several years working on using cryptography to secure distributed systems without widespread deployment of these ideas, leading at least one prominent researcher to muse publicly that this line of research, while theoretically intriguing, had been a practical failure. Within a few years, however, technology had reduced the costs of these techniques and applications had become more demanding—and rather suddenly distributed authentication was in wide use. Almost one decade after Castro and Liskov's seminal paper [6], we believe history is about to repeat itself.

In the same way that RAID disks are standard techniques for reliable storage despite the extra costs, BFT has the promise to become the norm in reliable systems. To make BFT the norm for deploying highly reliable and available systems, however, it is necessary to revisit how BFT systems are designed and implemented. We argue that optimizing the common case at the expense of introducing complicated corner cases and performance trapdoors in the presence of failures is going after fools gold. Fortunately, we believe that Aardvark demonstrates that one can have one's cake and eat it too—just hold the ice cream.

6 Acknowledgements

The authors would like to thank the LADIS PC for the comments and feedback. This work was partially supported by NSF grants CSR-PDOS-0509338 and CSR-PDOS-0720649.

References

[1] M. Abd-El-Malek, G. Ganger, G. Goodson, M. Reiter, and J. Wylie. Fault-scalable byzantine fault-tolerant services. In *Proc. 20th SOSP*, Oct. 2005.

[2] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth. BAR fault tolerance for cooperative services. In *Proc. 20th SOSP*, Oct. 2005.

[3] Y. Amir, B. Coan, J. Kirsch, and J. Lane. Byzantine replication under attack. In *DSN 2008*, 2008.

[4] W. Bartlett and L. Spainhower. Commercial fault tolerance: A tale of two systems. *IEEE Transactions on Dependable and Secure Computing*, 1(1):87–96, 2004.

[5] M. Burrows. The chubby lock service for loosely-coupled distributed systems. In *OSDI 2006*, pages 335–350, Berkeley, CA, USA, 2006. USENIX Association.

[6] M. Castro and B. Liskov. Practical Byzantine fault tolerance. In *Proc. 3rd OSDI*, pages 173–186, Feb. 1999.

[7] M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 2002.

[8] At lax, computer glitch delays 20,000 passengers. http://travel.latimes.com/articles/la-trw-lax12aug12.

[9] A. Clement, M. Marchetti, E. Wong, L. Alvisi, and M. Dahlin. Making Byzantine fault tolerant systems tolerate Byzantine faults. In *OSDI*, 2009.

[10] Contingency planning, for technology and terrorism. http://en.wikipedia.org/wiki/HurricaneKatrina.

[11] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira. HQ replication: A hybrid quorum protocol for Byzantine fault tolerance. In *Proc. 7th OSDI*, Nov. 2006.

[12] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: amazon's highly available key-value store. *SIGOPS Oper. Syst. Rev.*, 41(6):205–220, 2007.

[13] S. Ghemawat, H. Gobioff, and S.-T. Leung. The google file system. In *Proc. 19th SOSP*, pages 29–43. ACM Press, 2003.

[14] J. Gray. A census of Tandem system availability between 1985 and 1990. *IEEE Trans. on Reliability*, 39(4), Oct. 2000.

[15] J. Hendricks, G. R. Ganger, and M. K. Reiter. Low-overhead byzantine fault-tolerant storage. In *SOSP*, 2007.

[16] C. Ho, R. van Renesse, M. Bickford, and D. Dolev. Nysiad: Practical protocol transformation to tolerate byzantine failures. In *NSDI*, 2008.

[17] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: speculative byzantine fault tolerance. In *SOSP*, 2007.

[18] R. Kotla and M. Dahlin. High throughput Byzantine fault tolerance. In *DSN*, June 2004.

[19] S. S. Mukherjee, J. S. Emer, and S. K. Reinhardt. The soft error problem: An architectural perspective. In *HPCA*, pages 243–247, 2005.

[20] Hurricane Katrina. http://en.wikipedia.org/wiki/HurricaneKatrina.

[21] D. Oppenheimer, A. Ganapathi, and D. Patterson. Why do internet services fail, and what can be done about it, 2003.

[22] V. Prabhakaran, L. N. Bairavasundaram, N. Agrawal, H. S. Gunawi, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Iron file systems. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*, pages 206–220, New York, NY, USA, 2005. ACM Press.

[23] P. Shivakumar, M. Kistler, S. W. Keckler, D. Burger, and L. Alvisi. Modeling the effect of technology trends on the soft error rate of combinational logic. In *DSN '02: Proceedings of the 2002 International Conference on Dependable Systems and Networks*, pages 389–398, Washington, DC, USA, 2002. IEEE Computer Society.

[24] A. Sing, T. Das, P. Maniatis, P. Druschel, and T. Roscoe. Bft protocols under fire. In *NSDI*, 2008.

[25] J. Yang, C. Sar, and D. Engler. Explode: a lightweight, general system for finding serious storage system errors. In *USENIX'06: Proceedings of the 7th conference on USENIX Symposium on Operating Systems Design and Implementation*, pages 10–10, Berkeley, CA, USA, 2006. USENIX Association.

[26] J. Yang, P. Twohey, D. Engler, and M. Musuvathi. Using model checking to find serious file system errors. In *OSDI'04: Proceedings of the 6th conference on Symposium on Opearting Systems Design & Implementation*, pages 19–19, Berkeley, CA, USA, 2004. USENIX Association.

[27] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for Byzantine fault tolerant services. In *Proc. 19th SOSP*, 2003.