Conflict-free Quorum-based BFT Protocols

Atul Singh[†◇] Petros Maniatis[‡] Peter Druschel[◇] Timothy Roscoe[*]

Rice University[†] Intel Research Berkeley[‡] MPI-SWS[◇] ETH Zürich[*]

Technical Report MPI-SWS-2007-001

Abstract

Quorum-based Byzantine fault-tolerant protocols for replicated state machines allow replicas to respond to client requests without explicitly agreeing on the request ordering. As long as concurrent write operations do not conflict, quorum-based protocols are more efficient than agreement-based protocols. However, resolving conflicting writes and bringing replicas up-to-date with each other is a principal performance limitation of existing quorum protocols. We present a simple technique based on an un-trusted pre-serializer to *completely* mask such quorum-based BFT protocols from experiencing conflicting writes. Experimental results show that a non-faulty pre-serializer enables such quorum protocols to retain their efficiency even under significant write contention.

1 Introduction

Byzantine Fault-Tolerant (BFT) protocols have received considerable attention in the systems research community of late, because of their useful (and provable) correctness properties combined with a strong adversarial model. With BFT replicated state machines, programmers can write sequential, fault-oblivious code that implements a server state machine. The protocol then ensures that the replicated server state machine (RSM) executes requests sequentially and atomically in the order clients submitted them (linearizability), that it makes progress despite transient network faults (liveness), and that it can mask a bounded number of arbitrary replica failures.

There are two well-known classes of Byzantine fault-tolerant protocols: agreement-based and quorum-based. Agreement-based protocols, such as the well-studied PBFT [3] protocol, require replicas to first agree on a unique, serial ordering of requests; the requests are then executed in that order and replies send to the clients. Quorum-based protocols, on the other hand, are *optimistic* since they do not require replicas to explicitly run an agreement phase. Typically, quorum-based protocols require higher replication ($5f + 1$ compared to $3f + 1$ for agreement-based) but are more efficient since they can complete requests in one to two communication rounds compared to three for agreement-based protocols. However, conflicting write operations – write operations that cause replicas to become inconsistent with each other since they are received by different replicas in different order – causes severe performance degradation of these quorum protocols. The performance of agreement-based protocols is not vulnerable to such conflicting writes and we aim to provide similar robustness in quorum-based protocols.

We focus on two state-of-the-art quorum-based BFT RSM protocols: the Q/U protocol [1] and the HQ protocol [4]. Q/U requires $5f + 1$ replicas and completes requests in a single communication round between client and replicas when writes do not conflict. HQ requires $3f + 1$ replicas and completes requests in two communication rounds when there are no write conflicts. Detecting and resolving inconsistent replica states due to conflicting write operations is a costly operation in both protocols – Q/U requires exponential back-off from clients, while HQ invokes an agreement protocol that requires three extra phases to resolve conflicts. Consequently, performance of these protocols severely degrades under write contention.

In this paper, we present a simple technique to avoid write contention altogether from the existing quorum based BFT RSM protocols. We introduce a *pre-serializer*

node, which sequences client requests before they are handled by the quorum system. As long as the pre-serializer remains correct, replicas never experience conflicting writes on the same object from multiple clients and so never have to resolve such write conflicts. A correct pre-serializer not only avoids conflict resolution but also enables *batching*, a powerful optimization where multiple requests are processed in a single protocol execution. A faulty pre-serializer, on the other hand, can at worst introduce write conflicts in the system – a situation already handled by the base quorum protocol. To ensure that the pre-serializer is a non-faulty node most of the time, we also present a simple mechanism to switch the pre-serializer if it is suspected faulty by a sufficient number of other replicas.

To demonstrate the benefits of pre-serialization, we apply it to the HQ protocol and evaluate it experimentally. Our results demonstrate that pre-serialization achieves non-trivial improvements: for example, at $f = 3$, we achieve a factor of two improvement in throughput while reducing the network traffic by a factor of seven when compared to the original HQ protocol.

The rest of the paper is organized as follows. We start by giving a brief background on existing BFT RSM protocols in Section 2. In Section 3, we describe in detail how pre-serialization works in the context of HQ protocol and briefly sketch the design for pre-serialization in the Q/U protocol. We present experimental results in Section 4, discuss related work in Section 5, and conclude in Section 7.

2 Background

2.1 Q/U

Query/Update or Q/U [1] is an example of a single-round quorum-based protocol that tolerates up to f faulty replicas within a set of $5f + 1$ replicas. Replicas optimistically execute requests locally without explicitly agreeing on a common order. Clients cache replica histories present in the responses and append them to future requests. When an ordering conflict is detected among replicas – i.e., replica histories do not match in the responses – clients notify the replicas of the inconsistency and drive the system toward a consistent state. Typically, conflicting requests are aborted and retried using an exponential back-off mechanism once replicas have become consistent again.

Q/U requires a significantly lower number of messages and fewer rounds than PBFT, but can be mired by livelock under high-concurrency workloads, where multiple clients issue requests to the RSM concurrently.

2.2 HQ

To deal with the performance limitations of PBFT and Q/U, Cowling et al.'s HQ protocol [4] combines the quorum and consensus approaches in an ingenious way. HQ is a two-round quorum protocol in the absence of conflicting write requests (a common case) and requires $3f + 1$ replicas. Replicas optimistically choose an ordering of request (a *grant*) and notify the client. Client collect a quorum of $2f + 1$ grants and participate in a second round where they send back the collected grants (a *writeback*). Replicas can detect contention by observing the set of grants in the writeback message. Upon detecting contention, replicas resorts to Byzantine consensus for efficient conflict resolution. The relatively expensive consensus is only used to resolve concurrency conflicts, which would require exponential back-off in a pure quorum system such as Q/U. As a result, HQ improves significantly upon PBFT in low-concurrency settings, while resolving concurrency conflicts at a lower expected latency than Q/U.

HQ's conflict resolution technique relies on utilizing a *proxy server*. The proxy collects conflicting requests opportunistically, combines them into a conflict resolution batch, and submits them to the PBFT module (described below) for linearization. Once the order of conflicting requests has been agreed upon, the replicas validate the conflict batch: they check whether a conflict resolution was really needed, and whether all conflicting requests are valid. Then, they either execute the valid batch of conflicting requests in a consistent order, or otherwise reject the batch, replace the faulty proxy server that produced the batch via a PBFT view change, and try again. We now give a very brief overview of the PBFT protocol.

2.3 PBFT

The Practical Byzantine Fault Tolerant (PBFT) protocol [3] and its derivatives use 3-round Byzantine consensus over $3f + 1$ replicas to ensure that all clients' requests are executed in a consistent order in the face of up to f replica faults. PBFT requires a quadratic number of messages in the number of replicas and three rounds of message exchanges. While practical for small values of f, this limits scalability to large replica groups. The message complexity also leads to low throughput in bandwidth-constrained environments, and causes high request latency when the network delay among replicas is high.

3 Pre-serialized HQ (PS-HQ)

We now describe how pre-serialization works in the context of the HQ protocol. Our goal is to ensure that HQ does not experience write contention and therefore does not need to pay the price of conflict resolution. To that end, we interpose a pre-serializer node between the clients and replicas of the HQ protocol. Figure 1 presents the HQ protocols and the PS-HQ protocol. HQ completes requests in 4 message delays when there is no contention. PS-HQ completes requests in 5 message delays (extra phase due to pre-serialization) when pre-serialization is correct and irrespective of the write contention in the workload. We believe that the additional penalty of one message delay imposed by pre-serialization is easily outweighed by the benefits realized in the common case when the pre-serializer is correct.

Next we present the PS-HQ protocol in detail.

3.1 Design

PS-HQ clients behave as with HQ.

Pre-serialization is done by one of the replicas: at any time, it is the i-th replica, $i \equiv s \mod N$, where s is the current sequence number of the PBFT protocol used for conflict resolution, and $N = 3f + 1$ is the number of replicas[1].

[1]In general, pre-serializer could be from the client population, or be an entirely separate node.

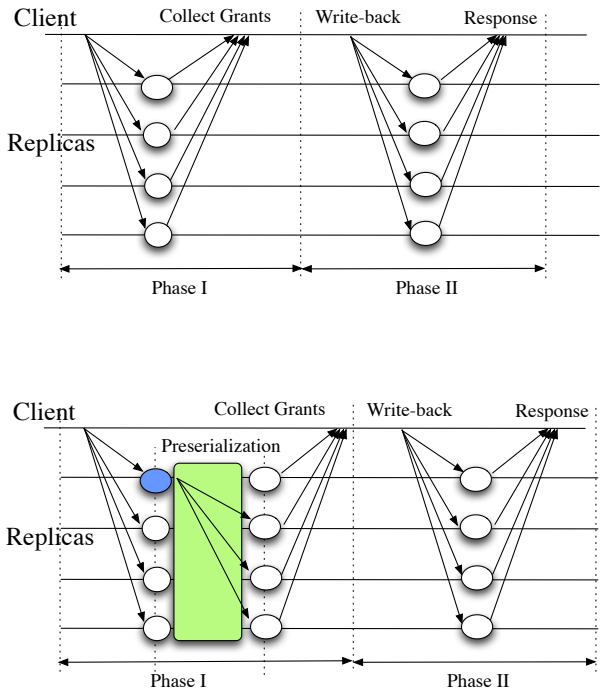

Figure 1: **HQ protocol (top) and PS-HQ protocol (below). Note that PS-HQ incurs extra message delay to send the pre-serialization messages from the pre-serializer (replica colored blue) to other replicas.**

The pre-serializer buffers client requests and submits them to the HQ replicas, serially, like a regular HQ client. Once it has received a quorum of $2f+1$ acknowledgments (signed hashes of the request) from distinct HQ replicas, it can submit the next serialized client request.

Otherwise, PS-HQ replicas behave identically to HQ replicas with one exception: they do not process regular clients' requests immediately, but associate a timer with each one and wait for the same request to be received from the pre-serializer. If the request is received in time from the pre-serializer, the replica cancels the timer, sends a grant to the client and an acknowledgment to the pre-serializer. If the timer for a client's request expires before that request is received from the pre-serializer, the replica initiates a pre-serializer change.

A client request that arrives at a replica after it was

received from the pre-serializer is ignored. If a replica received a request different from the request contained in the pre-serializer message and both are signed by the client and have the same RID (an RID is assigned by the client to every request it sends), it generates a grant only for the pre-serialized request and drops the other request.

Replicas handle pre-serializer changes in the same way HQ handles conflict resolution. A replica sends a START message that may contain a conflict certificate (if the faulty pre-serializer submitted conflicting writes to different replicas) or not (if the pre-serializer suppressed a client request causing timers to expire). Conflict resolution eventually results in a PBFT invocation, which increments the PBFT sequence number and resolves any conflicting writes. Note that incrementing the PBFT sequence number changes the pre-serializer. Hence, the cost of changing the pre-serializer is similar to the cost of resolving conflicts in HQ.

3.2 Correctness

A faulty pre-serializer appears to HQ replicas as a faulty client that submits different requests to different replicas. As a result, PS-HQ shares HQ's safety properties, by relying on HQ's tolerance of an arbitrary number of faulty clients. This also holds when a faulty pre-serializer colludes with faulty clients.

Moreover, while a faulty pre-serializer may suppress or delay a correct client's request or batch of requests, this behavior eventually results in a pre-serializer change. Therefore, progress is ensured as long as clients retransmit their requests until they succeed.

Now we argue, informally, that PS-HQ's performance is more robust to faulty clients compared to HQ when the pre-serializer is non-faulty.

A faulty client cannot force PS-HQ into conflict resolution by submitting different requests with the same RID to different replicas. This is because in PS-HQ, requests are not given a grant at the replicas until they are pre-serialized. A non-faulty pre-serializer handles only the first request that it receives with a given RID from a client. A non-faulty replica gives a grant only to the request identified in the pre-serialization message. Recall that a non-faulty replica drops other requests that have the same RID and are signed by the same client. Therefore, a faulty client cannot introduce write conflicts as long as the pre-serializer is non-faulty. Also, ignoring such requests ensures that faulty clients alone cannot cause a pre-serializer change.

A faulty client may, however, delay writes since PS-HQ relies on clients to perform the second phase of the write protocol. In HQ, if replicas receive a new write request for an object for which there is a pending write (this happens when a replica is waiting to receive the 2nd phase message from the client), they send a refusal response to the new request, prompting the new client to complete the previous request. We use the same approach to handle faulty clients in PS-HQ.

3.3 Batching

In the PS-HQ protocol as presented so far, the pre-serializer forwards every single client request to the HQ replicas individually. In addition, the existence of a single pre-serializer enables *batching*: instead of sending individual requests, the pre-serializer can combine multiple client requests into a single batch, thus amortizing the cost of generating authenticators and the bandwidth of the authenticators themselves over many requests.

Batching does not affect the structure of the protocol but only the number of individual messages sent: upon receiving a batch, replicas create a cumulative grant for the entire batch and send it to a client chosen deterministically from the batch (e.g., the owner of the first request in the batch). A non-faulty client creates a write certificate for the batch and sends it again to the replicas in phase 2. The replicas then send a reply message for every request in the batch to the client who issued that request. Note that replicas create a single grant for the entire batch. As a result, they save both processing (by computing a single authenticator rather than one for each included request) and bandwidth (by sending a single grant rather than one for each included request) proportional to the batch size.

3.4 Expected Improvements

Preserialization is based on the premise that conflict resolution is an inherently expensive task. Preserialization replaces the cost of conflict resolution with the cost of interposing a sequencer between clients and replicas that – if non-faulty – serializes the request stream to remove

all sources of write contention. The cost of conflict resolution varies among quorum systems; for HQ, it is the cost of linearizing a conflict resolution request via a PBFT module, which adds several replica-to-replica communication rounds, additional authenticator computations, and some state. In Q/U, it is the latency cost inherent in exponential back-off, as well as the additional bandwidth and computation for protocol message retransmissions.

The request batching performed by preserialization offers further benefits, since it replaces the costs of computing and transmitting individual authenticators for multiple requests with those for a single, larger request. In HQ's case, the conflict resolution mechanism applies to multiple (conflicting) requests in a single PBFT invocation. Therefore, preserialization without batching for HQ may actually *reduce* performance under contention, since it causes every request to go through its own protocol session (two rounds of client-replica exchanges), whereas HQ's conflict resolution might handle them as part of a batch.

However, PS-HQ batching goes beyond what HQ's conflict resolution batching can accomplish, for two reasons. First, with regards to conflicting requests for the same object, PS-HQ can get the same batching benefits as HQ's conflict resolution, but at a much lower cost: a single grant and write-2 message exchange for an entire batch, and no additional PBFT protocol invocation over the entire batch. Second, PS-HQ also batches *non-conflicting* requests, e.g., requests for disparate objects. Therefore, PS-HQ improves performance even when write contention is low but there is a high rate of requests to different objects.

Overall, at very low contention and in the absence of sustained load, we expect HQ to be superior to PS-HQ due to the overhead of preserialization. As sustained load increases, we expect PS-HQ to be competitive with HQ at low contention. At high contention PS-HQ should dominate HQ, since PS-HQ's throughput is not affected by contention.

In the presence of faults, PS-HQ's performance should be roughly similar to HQ. PS-HQ initiates a pre-serializer change, which itself involves conflict resolution, whereas HQ simply performs conflict resolution.

In Section 4, we confirm these intuitive performance benefits of PS-HQ experimentally.

3.5 Pre-serialized Q/U (PS-Q/U)

We present here only an outline of how pre-serialization works in the Q/U protocol since most of the logic and correctness arguments are similar to PS-HQ.

We introduce a counter variable in the Q/U replica state to count the number of invocations of conflict resolution and denote it by C_{CR}. At each conflict resolution, this counter is incremented. Since replicas synchronize during conflict resolution, C_{CR} is also synchronized. Note that C_{CR} serves the same role as the sequence number s maintained by the PBFT module in PS-HQ.

At a given C_{CR}, a replica with id $i \equiv C_{CR} \mod N$ serves the role of the pre-serializer.

The client logic remains unchanged.

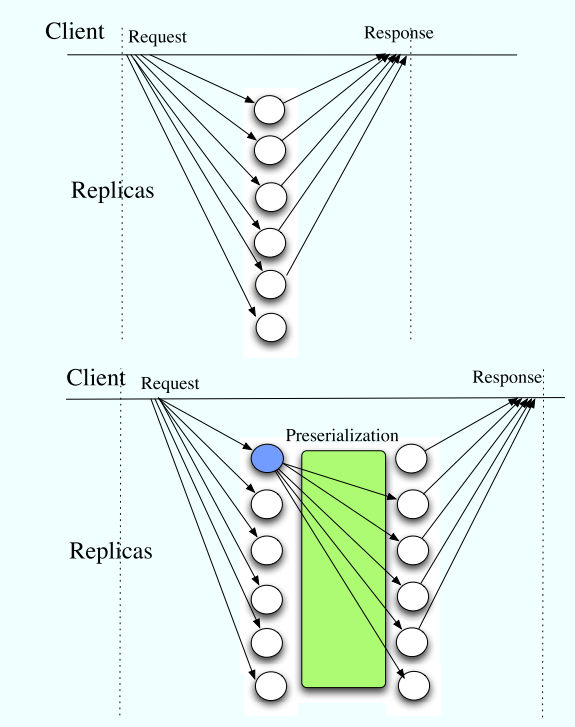

Figure 2: **Pre-serialization in Q/U protocol.**

The pre-serializer serializes requests in a fashion similar to PS-HQ. When the pre-serializer is non-faulty, requests complete in three message delays between client and replicas (shown in Figure 2) compared to two mes-

sage delays required by Q/U when there are no conflicts. We believe that this extra cost due to pre-serialization is more than offset by the advantages offered by pre-serialization in the face of high write concurrency and low fault incidence.

If the pre-serializer is faulty, it can at worst cause writes to conflict. Since Q/U already handles conflicts (at the potentially high cost of exponential back-off mechanism), we relegate dealing with faulty pre-serializers to that mechanism. Also, note that C_{CR} increments during conflict resolution, so a new pre-serializer is chosen after every conflict resolution.

The authors of Q/U observed that Q/U exhibits a drop in throughput as f grows. This is due to the use of authenticators, whose cost of generation and verification grows linearly with f. Pre-serialization can mitigate some of these problems via batching – multiple requests are performed at the same logical timestamp. This reduces the computational load at the pre-serializer and also enables multiple requests to share the same authenticator, i.e., one authenticator is generated over a batch of requests.

4 Experimental Evaluation

We evaluate the effectiveness of preserialization by implementing it in the HQ codebase, which was provided to us by the HQ authors. We are unable to evaluate preserialization with the Q/U protocol since its implementation was not publicly available at the time of this writing.

In this section, we show that in the absence of faults and as write contention increases, PS-HQ maintains a roughly constant (1) high throughput, (2) low bandwidth usage, and (3) low latency. In contrast, HQ's throughput decreases, its bandwidth use increases, and its request latency increases as contention grows.

We used Emulab for our experiments. We ran experiments on 46 "pc3000" machines, each having a 3 GHz processor with 2 GBytes of RAM and connected via a 1000 Mbps router. Nodes were connected in a virtual LAN. We ran server code on 4 to 16 machines (for the number of tolerated faults ranging from $f = 1$ to $f = 5$). Four clients each were running on 30 machines with very low utilization, for a total of 120 clients.

For a fair comparison, we used the same synthetic workload as that used in the original HQ study. Each client has a private object (which no other client ever writes and, as a result, never experiences any write contention). There is also a single object globally shared by all clients. Each client has only one outstanding request at any given time; when it receives a response to its previous request, it picks an object (its private object or the shared one) and it generates a new request for it. The choice of whether to send a request to the shared or the private object is governed by a *contention parameter*: the probability that a client will choose the shared object when sending a new request. When that parameter is set to 0, no request is sent to the shared object and there is no write contention. When that parameter is set to 1, all requests are sent to the shared object, resulting in 100% contention.

For each combination of replica group size (4, 7, 10) and contention parameter setting (from 0 to 1), we show the throughput in requests per second (computed by timing the execution of $100,000$ requests), the bandwidth used in KBytes per request (computed by counting all bytes sent and dividing by the number of requests), and the request latency in seconds (by measuring the average amount of time it takes for a request to receive a response).

Figure 3 compares PS-HQ to HQ.

At low contention, the throughput improvement is moderate, primarily thanks to multi-object batching. At high contention, the improvement is more pronounced and grows as the size of the replica group grows, from a little under a factor of two for group size 4 to almost a factor of 2.5 for group size 10.

HQ's write throughput is always inferior compared to PS-HQ's.

5 Related work

Our work falls in the space of optimization techniques to make Byzantine fault tolerant protocols more practical.

Castro and Liskov [3] presented PBFT, a BFT algorithm that is suitable for Internet applications. Even though PBFT is more efficient than earlier proposals, its authors recognized the high per-request overhead and introduced three powerful optimizations: batching, pipelining, and tentative execution. Unfortunately, as f increases, the benefits of these optimizations can not overcome the fundamental bottleneck of PBFT: a quadratic

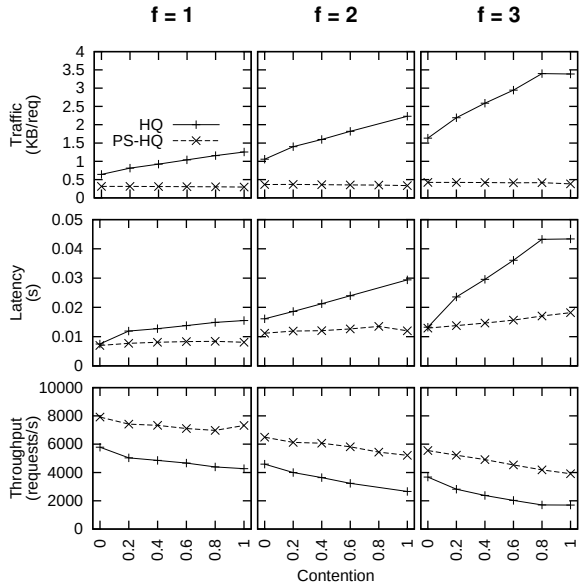

Figure 3: Throughput (bottom) measured in completed requests per second, per request latency (middle) measured in seconds, and traffic (top) measured in KBytes per request. Contention is ranging from 0 to 100% on the x axes. f is ranging from 1 to 3 from left column of graphs to right, corresponding to replica group sizes of 4, 7, and 10, respectively.

number of message exchanges in the number of replicas to reach agreement.

Yin et al. [10] made a clever observation that by separating the agreement phase from the execution phase of the PBFT protocol, the hardware cost of replication can be reduced. This separation allows different machines for the agreement and execution phases, allowing different services to share the same agreement replicas. At the same time, only $2f + 1$ replicas are required (f fewer than with PBFT) for the execution phase once requests are ordered by the agreement replicas.

Several optimizations have been proposed that attempt to reduce the cost of consensus when faults are rare. Fast-Paxos [2] can deliver consensus in two communication rounds when there are no faults and the network is synchronous, assuming crash-only faults. Under no faults and perfect synchrony, Kursawe [6] reduced the cost of Byzantine consensus by completing in two rounds and requiring only $3f + 1$ replicas. In less favorable settings, Kursawe's protocol falls back to the traditional three round protocol. Martin and Alvisi [8] proposed a Byzantine consensus protocol that requires $5f + 1$ replicas and always completes in two rounds except in the case where the leader is faulty. Our technique is also based on the idea of optimizing the fault-free case but we apply it in the context of quorum-based BFT protocols to avoid conflicting writes completely.

Li and Mazières [7] extend the fault tolerance of BFT protocols by providing fork* consistency (a form of consistency weaker than linearizability) when there are more than f faults in the system. Rodrigues et al. [9] also improve the fault scalability of BFT protocols without relaxing the consistency, but by identifying a fundamental tradeoff between safety and liveness; for example, they show that the PBFT protocol can be configured to be safe even if 2/3-rds of the replicas are faulty, but loses liveness as soon as 1/6-th of the replicas are faulty.

More recently, Kotla et al. present Zyzzyva [5], a PBFT variant, that exploits speculation extensively to reduce the overhead and latency of BFT replication. In failure-free and synchronous executions, Zyzzyva is extremely efficient since requests complete in 3 message delays. Unfortunately, under slightly worse conditions such as a single faulty replica or network jitter, Zyzzyva requires 5 message delays which is similar to our PS-HQ protocol. To avoid the additional message delays, the authors propose Zyzzyva5, which requires $5f + 1$ replicas and completes requests in 3 message delays even if there are f faults in the replicas (except the primary). Interestingly, Zyzzyva5 appears similar to PS-Q/U with PS-Q/U likely to be more robust to faulty clients since Q/U does not have a second phase. Finally, Zyzzyva's view change protocol is more heavy-weight and complex compared to PBFT's view change protocol, which is used in PS-HQ. PS-Q/U does not have a view change protocol.

6 Acknowledgment

We thank the authors of HQ, especially James Cowling, for sharing the HQ codebase with us and answering our queries regarding both the HQ protocol as well as the codebase. We also thank Lorenzo Alvisi for his helpful

comments.

7 Conclusion

We have presented a simple yet powerful optimization technique for state-of-the-art quorum-based BFT protocols. Results show that when faults are rare, our optimization delivers significant benefits.

References

[1] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. Reiter, and J. J. Wylie. Fault-scalable Byzantine fault-tolerant services. In *Proceedings of ACM Symposium on Operating System Principles (SOSP)*, Brighton, UK, Oct. 2005.

[2] R. Boichat, P. Dutta, S. Frolund, and R. Guerraoui. Reconstructing Paxos. In *SIGACT News*, 2003.

[3] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *Proceedings of USENIX Operating System Design and Implementation (OSDI)*, New Orleans, USA, Feb. 1999.

[4] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira. HQ Replication: A Hybird Quorum Protocol for Byzantine Fault Tolerance. In *Proceedings of USENIX Operating System Design and Implementation (OSDI)*, Seattle, USA, Nov. 2006.

[5] R. Kotla, L. Alvisi, M. Dahlin, A. Clement, and E. Wong. Zyzzyva: Speculative Byzantine Fault Tolerance. In *Proceedings of ACM Symposium on Operating System Principles (SOSP)*, WA, USA, Oct. 2007.

[6] K. Kursawe. Optimistic Byzantine Agreement. In *Proceedings of IEEE Symposium on Reliable Distributed Systems (SRDS)*, Suita, Japan, Oct. 2002.

[7] J. Li and D. Mazières. Beyond One-third Faulty Replicas in Byzantine Fault Tolerant Systems. In *Proceedings of USENIX Networked Systems Design and Implementation (NSDI)*, Boston, MA, USA, Apr. 2007.

[8] J. Martin and L. Alvisi. Fast Byzantine consensus. In *Proceedings of International Conference on Dependable Systems and Networks (DSN)*, Yokohama, Japan, June 2005.

[9] R. Rodrigues, P. Kouznetsov, and B. Bhattacharjee. Large-Scale Byzantine Fault Tolerance: Safe but Not Always Live. In *Proceedings of Hot Topics in Dependability (HotDep)*, Edinburgh, UK, June 2007.

[10] J. Yin, J.-P. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for byzantine fault tolerant services. In *Proceedings of ACM Symposium on Operating System Principles (SOSP)*, pages 253–267, Bolton Landing, NY, USA, Oct. 2003.