Large-Scale Byzantine Fault Tolerance: Safe but Not Always Live

Rodrigo Rodrigues
INESC-ID and
Technical University of Lisbon

Petr Kouznetsov
Max Planck Institute
for Software Systems

Bobby Bhattacharjee
University of Maryland

Abstract

The overall correctness of large-scale systems composed of many groups of replicas executing BFT protocols scales poorly with the number of groups. This is because the probability of at least one group being compromised (more than $1/3$ faulty replicas) increases rapidly as the number of groups increases. In this paper we address this problem with a simple modification to Castro and Liskov's BFT replication that allows for arbitrary choice of n (number of replicas) and f (failure threshold). The price to pay is a more restrictive liveness requirement, and we present the design of a large-scale BFT replicated system that obviates this problem.

1 Introduction

Byzantine-fault-tolerant replication is a popular tool to ensure correct and highly-available operation of important services when individual components may fail arbitrarily (e.g., due to intrusions or software errors).

Numerous Byzantine-fault-tolerant (BFT) replication protocols have been proposed [18, 5, 1, 6], and more recently these protocols have been used as building blocks to develop large-scale systems like Farsite [2], OceanStore [11], or Rosebud [20]. These large-scale systems are usually organized as follows. The system state is split into a number of *objects* and each object is managed by a separate state machine replication group, with varying assignment strategies. Each state machine replication group is implemented by a small subset of the system members which create the illusion of the object being hosted by a single correct and available server.

A problem that arises in such systems is that, assuming some fixed fraction of faulty nodes in the system, the probability of a single replica group exceeding its failure threshold (usually $1/3$ of faulty nodes), and therefore work incorrectly, increases rapidly as the number of groups in the system increases. Once the failure threshold is exceeded in a single group, the correctness of the system as a whole might be compromised.

This problem is illustrated by Figure 1 that shows how the probability of having at least one group exceeding its failure threshold varies with the number of groups in the system. The different curves represent different fractions of faulty nodes in the system and values of f (where $n = 3f + 1$ is the number of replicas in each group). This shows that even with 1% faulty nodes and $f = 2$ there is a reasonable chance that one of the groups will not meet its correctness condition when the number of groups in the system goes beyond 100.

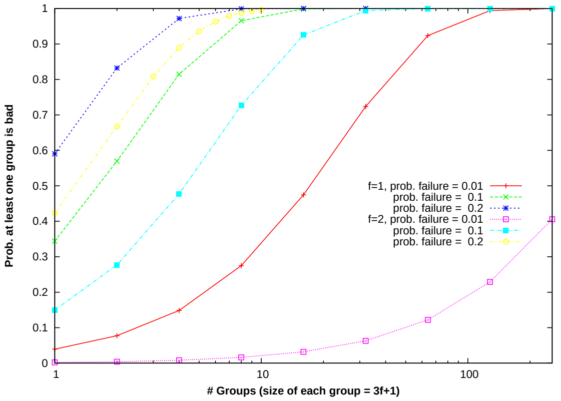

Figure 1. Probability of the existence of faulty groups, as a function of the number of groups, for different fractions of faulty nodes in the system.

This problem is exacerbated in systems where the fraction of faulty nodes is relatively high, even if that only occurs for a short period of time. For instance, during the early stages of a worm outbreak the fraction of nodes that do not behave correctly may increase substantially for some time, even beyond $1/3$ of the system nodes, which would cause most of the replication groups that implemented the above protocols to break, even if we configured the system with a large value of f.

One of the ways to overcome this issue is to modify BFT replication protocols to relax the consistency criteria of the system. E.g., the system described in [13] is able to tolerate up to $2/3$ of faulty replicas in a BFT group by providing weaker "fork" semantics.

This paper proposes an alternative approach based on the following principles. First, we present a modification to Castro-Liskov's Practical Byzantine Fault Tolerance Protocol (CLBFT) that allows it to tolerate any number faults in the replica group up all but one replica, without sacrificing the semantics provided by the protocol.

The problem with this modified version of CLBFT is that it has an increased chance of running into liveness problems. In this paper we also present two strategies to address this new problem in the context of a large-scale system with multiple replication groups.

The first strategy is to treat each modified CLBFT group as a node that executes a replication protocol tolerant to fail-stop (silent) failures. This strategy has some limitations, namely in terms of its performance, so we also present the design of a system called ShowByz that overcomes the liveness problems of individual groups. ShowByz assigns two groups to each subset of the state, a primary and a backup group. Periodically the backup group checks the liveness of the primary group and transfers the missing state from that group. If the backup group detects problems then it takes over the responsibility, allowing operations to proceed despite liveness problems in one of the groups. We also show how to address issues like running operations efficiently (without the need of contacting multiple replica groups), and yet dealing with the possible loss of operations that executed in a group that subsequently halted.

2 Modified CLBFT

One of the key aspects of ShowByz is the use of a modified version of Castro-Liskov's BFT state machine replication protocol [5]. This simple modification will enable the protocol to have improved safety properties (in particular, overcoming the barrier of requiring less than $1/3$ faulty nodes to provide correct behavior), but at the expense of sacrificing liveness, even with a small fraction of faulty nodes.

The modification is based on the simple observation that CLBFT's operations are justified by certificates. These are matching statements authenticated by a *quorum* of $2f + 1$ out of the $3f + 1$ system nodes.[1] The key to the safety properties of CLBFT is that these quorums that form the certificates intersect in at least one non-faulty replica, and non-faulty replicas never perform actions that would break the safety properties of the system (in particular by assigning the same sequence number to different operations).

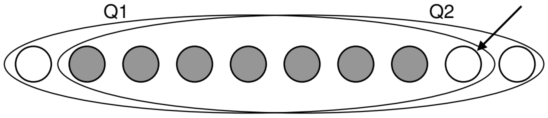

Figure 2. A system with 10 replicas, 7 of which are faulty (represented by a shaded circle), and two quorums, Q1 and Q2, which intersect in a single non-faulty replica (marked by an arrow).

This explains why CLBFT requires $3f + 1$ replicas to achieve both safety and liveness: Assuming a total of n replicas, the algorithm must proceed after communicating with a quorum of $n - f$ replicas, since f replicas might be faulty and not respond. However, the quorum intersection properties must ensure that any two quorums intersect in at least one non-faulty replica. Since the size of the intersection between two quorums is $n-2f$, this yields $n-2f > f$, or $n > 3f$.

Our insight is to increase the quorum size (in proportion to the number of replicas), sacrificing liveness, but preserving safety by maintaining the quorum intersection properties.

In particular, we can use a quorum size of $Q = \lfloor \frac{n+f}{2} \rfloor + 1$ for an arbitrary choice of f and n (with the only restriction that $n > f$). For instance, if we pick $n = 10$ and $f = 7$, this yields a quorum size of 9 replicas, as depicted in Figure 2. This way, any two quorums will still intersect in one non-faulty replica, which guarantees safety despite a large fraction (in this case 70%) of faulty replicas. The downside is that there are liveness problems when two or more replicas (20%) are unavailable, since we cannot form a quorum of seven replicas at that point. In the next sections we show how a large-scale system that uses modified CLBFT can address the liveness issues.

3 Large-Scale BFT Systems

The modifications to CLBFT presented in the last section enable us to overcome the main problem raised by large-scale systems that have multiple BFT groups (namely, that the likelihood of one of the groups overcoming the $1/3$ faulty nodes barrier increases rapidly with the size of the system). Unfortunately, this is done at the expense of increasing the likelihood of having liveness problems, so we need to address this issue.

[1] Occasionally there are certificates with $f + 1$ nodes, whenever the presence of a single non-faulty node suffices.

In this section we explore two different strategies for building systems based on multiple groups that use the modified CLBFT from the previous section.

3.1 Objectives

The overall goal of the system design we will present in this section is build a system that is composed of multiple objects, where different objects may be implemented by different replica groups.

Clients can perform arbitrary operations on any object, just like they would on a single state machine replication group, with the only difference that they have to identify the object where the operation is going to run in the library call that invokes the operation.

The semantics provided by our system is linearizability [9] of operations on individual objects. This means operations in a particular object appear to execute in some sequential order that is consistent with the real-time order in which the operations are executed. Our current design does not support multi-object operations, this is left as future work.

3.2 Assumptions

We assume an asynchronous distributed system with a large number of nodes where the system membership might constantly change.

We assume the existence of a *configuration manager*, an oracle that can provide any system node with the current *system configuration* which consists of the current set of system members, the respective addresses and public keys, a partitioning of the service state into objects, and a an assignment of one or more replica groups for each object.

In practice, the system configuration may not have to include the partitioning of the system state and group assignments, since these may be implicit as happens in assignment schemes like consistent hashing [10].

In our final system design, we intend to implement this oracle as a BFT state-machine group of highly safe nodes, but details are left as future work.

To ensure safety, we assume that the correctness condition of each modified CLBFT replication group is met, i.e., its (possibly large) failure threshold f is not exceeded. For now we will assume that this is true all the time about every group that is formed in any configuration. In the future we intend to relax this assumption to state that this failure threshold cannot be exceeded during a time window of vulnerability, and define that window in terms of the occurrence of certain events.

For liveness we require eventual synchrony (as CLBFT also requires [5]), but we must also assume that "enough" modified CLBFT replica groups will make progress "while they are needed". After we explain our system design will restate this more precisely.

3.3 Alternative 1: Higher-Level Fail-Stop Replication

Our first attempt at a system design is to consider each modified CLBFT group to be a node that participates in a (higher level) replication protocol that tolerates fail-stop failures (e.g., Paxos [12]). The idea of reducing Byzantine behavior to fail-stop faults using replication, and then using a protocol that tolerates fail-stop nodes was originally proposed by Schlichting and Schneider [21], and was more recently adopted by Steward [3] in the same way we propose of composing CLBFT with Paxos.

The main advantage of this algorithm lies in its simplicity, and proven correctness, given that the higher-level fail-stop replication protocols are well-studied.

However, this solution may pose some problems when deployed in a wide-area setting. If all replicas of the CLBFT groups are located in the same site as proposed in Steward [3], then there is an increased likelihood of correlated failures (e.g., if all replicas are managed by the same administrator who repeats the same mistake), which could cause a Byzantine fault of a participant of the higher level fail-stop replication protocol, which is enough to break safety.

If, on the other hand, replicas of each CLBFT group are located in different sites, then the latency of executing each operation is exacerbated by the fact that each step of the higher level replication protocol requires a lower-level CLBFT operation among distant replicas.

3.4 Alternative 2: ShowByz

To overcome the problems of liveness of individual replica groups without incurring a possible performance degradation, we introduce a new design called ShowByz. ShowByz is efficient since each operation only needs to be executed in a single modified CLBFT group in the normal case, yet it can recover from a situation where the modified CLBFT that executed the operation halts because of liveness problems.

In summary, the way we achieve this is using a primary-backup protocol among replica groups, where operations are executed *speculatively* at the primary, and the result only becomes definitive after the backup group has copied the new state that reflects the operation from the primary group. Speculative execution has been proposed in the context of other client-server systems [16], and even Byzantine-fault-tolerant replicated systems [17]; we just apply that idea in a different context of executing operations speculatively in a single Byzantine replication group, and not having to wait for the operation to be propagated to the backup group. As in these systems, clients checkpoint their state and continue their execution based on the predicted results. If the prediction is correct, the checkpoint is discarded. If the predicted result turns out to be incorrect, the client rolls back to a previous state and the operation is retried (for details see [16]). If one of the groups fails, which would cause the definitive execution to halt, a configuration change is required to pick new primary and backup groups and continue to make progress. Note that, even though the speculative executions are expected to be normally correct, the strong consistency conditions are guaranteed to hold only with respect to the definitive results [16].

In the remainder of this section we will present a preliminary design of ShowByz. Some details will be omitted due to space limitations or left as future work. We will begin by explaining how the system works in the normal case when all the groups are live and there are no configuration changes (Section 3.4.1); then we explain how the system reacts to liveness problems in individual groups (Section 3.4.2); we discuss how the system handles configurations changes (Section 3.4.3); and we conclude with a brief correctness argument in Section 3.4.4.

3.4.1 Normal Case

In the normal case clients start by invoking a modified CLBFT operation on the primary group, which executes through the normal CLBFT protocol [5], returning the reply of the operation to the client.

Periodically, the backup group checks the liveness of the primary group, and transfers a snapshot of the service state as part of the liveness check. We intend to take advantage of BFT's checkpointing protocol [5] to implement this. CLBFT takes a logical snapshot (called a checkpoint) of the service state periodically to enable bringing slow replica up-to-date. Our protocol will mark some of these checkpoints as being the ones that will be transferred to the backup group, in particular those that occur when approaching the end of the liveness check interval.

The liveness check is implemented by having each replica in the backup group individually acting as a client of the primary group, executing a special state machine operation, which also reads the latest snapshot of the service state. (For efficiency it reads only the subset of the state that has been modified since the last transferred checkpoint.) There is also a second special state machine operation that acknowledges that the first operation was concluded successfully by the backup replica.

When a quorum of backup replicas run the acknowledgment operation, the liveness check is complete.

3.4.2 Liveness Problems (Faulty Groups)

The key to addressing the liveness issues raised by our modifications to the CLBFT protocol is a takeover protocol, where, when the primary group fails (i.e., cannot execute operations due to liveness problems), the backup group takes over the responsibility for running (speculative) operations.

First we will explain what happens when the primary group fails, we will then discuss failures of the backup group.

Primary Group Liveness Failures

When the liveness check fails, the backup replicas exchange signed "takeover" messages.

Once any backup replica gathers a quorum of signed "takeover" messages, it runs a "takeover" operation on the backup group, and after that it can start processing normal client state machine requests.

This raises the problem that some operations may have been lost in the meanwhile, particularly those executed since the latest checkpoint that was successfully transferred.

As mentioned, we solve this using speculative execution at the client, a solution that was proposed recently in the context of other distributed systems [16]. Clients run the operations on the primary group tentatively, and run a second operation later to see if the result is definitive (i.e., to confirm that the state has been transferred to the backup group successfully). If the second operation fails, the client operation must contact the backup group and possibly rerun the operation if it hadn't been run yet.

Backup Group Liveness Failures

It could also happen that the liveness check fails because the backup does not execute enough takeover operations. This could have happened due to problems in backup replicas or due to communication problems, such situations are impossible to distinguish in an asynchronous setting.

In this case the primary group can do two things. Either it halts (assuming the worst-case scenario of a network partition where the backup group has taken over), or it optimistically continues to execute speculative operations, which eventually may have to be rolled back.

In either case the primary group should request a configuration change which will select new primary and backup groups, in order to minimize client wait time for the definitive results.

An important detail is that an operation that only ran on the backup group (after a takeover) is also considered tentative, so when the primary group fails it is also convenient (for performance reasons) to request a configuration change.

3.4.3 Configuration Changes

The mechanism for configuration changes is also similar to other state machine replication systems that support a dynamic system membership [19, 14].

When a new configuration is issued, it is propagated to all system nodes by the oracle that produces configurations (and also using a gossip protocol). We will describe a situation where a configuration change generates a change both in the primary and backup groups (since other scenarios would be optimizations of this one).

Once an old replica (either primary or backup) receives the new configuration it runs a special "stop" operation on the state machine. This causes the old group to stop accepting normal operations except for state transfer operations to the new group. New replicas from the primary group will transfer state from the old groups by executing special state transfer operations. These can be run on either group (though the primary group is preferred for performance reasons).

After the new primary group concludes state transfer, the secondary group transfers state from the primary group and normal operation resumes.

Clients that had tentative operations (operations that executed in only one of the groups) check if they were transferred by querying the new primary or backup group, and may need to rerun them in the new groups.

The fact that the primary group in the new configuration will read the state from the old groups means it can decide between conflicting operations, precluding a situation where the system state would diverge if there is a conflict between a primary and a backup group. In this case such divergent operations would always be considered tentative in the old configuration, and state transfer to the new configuration will determine which of the two different outcomes will prevail.

This mechanism for state transfer introduces a requirement for liveness of the system, which is that one of the groups (either the primary or the backup) in each configuration must remain live until state transfer to the new configuration concludes.

3.4.4 Correctness

We briefly sketch an argument for why our system is correct. Given our correctness conditions, we know that every CLBFT group in the system will meet its failure threshold, hence will follow its specification [5]. Therefore we can regard these as fail-stop components.

We need to argue that the definitive results meet the linearizability semantics (since we do not provide any guarantees on tentative results). This is true within a configuration, since, for a result to be considered definitive, this involves agreement among both groups, and each implements linearizable semantics [5].

Across different configurations this is also true for definitive results, which will be transferred to the new groups in the same order (since state transfer reads from either group, but the operation was executed in both). For results that were not considered definitive we need to consider two cases. The first case is when the object was transferred to the new groups. In this case, the client will retry the operation in one of the new groups and will see the correct linearization of its operation. The other case is if the object was not transferred, but the client will retry the operation until it succeeds in applying the operation in both groups of the same configuration, which will ensure linearizable semantics.

4 Related Work

There are some proposals for large-scale systems that tolerate Byzantine faults, e.g., Farsite [2], OceanStore [11], or Rosebud [20]. These systems share a common design feature that they use multiple groups executing Byzantine replication protocols to partition the load among the system members. Since each group runs the original CLBFT, they only tolerate up to $1/3$ faults in each replica group. Our work is complementary to these previous systems, in the sense that they could replace these Byzantine groups with our two-level construction presented in the design of ShowByz, thereby achieving a correctness condition that would tolerate a larger fraction of faulty nodes.

Secure peer-to-peer routing overlays [4, 8, 15] are also instances of large-scale systems that tolerate Byzantine faults. However, this research is rather orthogonal to our own, since the design of secure peer-to-peer systems differs from the pattern of multiple state machine groups. Furthermore, the design of ShowByz would probably not be adequate to a highly dynamic peer-to-peer deployment where nodes constantly join and leave the system.

The idea of using speculative execution in the context of a client-server system was originally proposed by Nightingale et al. [16]. Our work uses that idea, but applies in a slightly different way, by executing operations speculatively in a single Byzantine replication group, avoiding having to wait for the operation to be propagated to the backup Byzantine replication group.

The idea of reducing Byzantine behavior to fail-stop faults using replication, and then using a protocol that tolerates fail-stop nodes was originally proposed by Schlichting and Schneider [21], and was more recently adopted by Steward [3] in the same way we propose of composing CLBFT with Paxos. However, the main purpose in Steward was to reduce the latency of wide-area CLBFT operations, since each local area site runs CLBFT and different CLBFT groups from different sites run Paxos among themselves.

Douceur and Howell [7] also analyze the problem of dealing with faulty groups in a large-scale system like Farsite, but from a different perspective of isolating the effects of faulty groups instead of preventing their existence.

The work of Li and Mazières [13] mentions a possibility of changing the quorum size in a variant of the CLBFT protocol to trade liveness for safety. However, the main focus of this work is on a weaker form of consistency that can be achieved with the original quorum sizes ($2f+1$ in a system of $3f+1$ nodes) when up to $2f$ nodes can fail, without sacrificing liveness. In contrast, for arbitrary replica group size and upper bound on the number of faulty nodes we can set the quorum sizes that ensure linearizability, and furthermore our work focuses mainly on the subsequent step of addressing the liveness issue introduced by this modification.

5 Conclusions

In this paper we address the problem of large-scale BFT systems that contain many replica groups, where a $1/3$ bound on faulty replica is likely to be exceeded in some group. We present a new approach that provides better safety properties (beyond this $1/3$ bound) at the expense of liveness, and we present a preliminary design of a system that overcomes the liveness problems with a simple and efficient higher-level fail-stop replication protocol, integrated with speculative execution.

Even though some details are still left open, we believe our approach contains interesting insights that can be efficiently used in the design of large-scale state-machine replication systems.

References

[1] M. Abd-El-Malek, G. R. Ganger, G. R. Goodson, M. K. Reiter, and J. J. Wylie. Fault-scalable byzantine fault-tolerant services. In *SOSP '05: Proceedings of the twentieth ACM symposium on operating systems principles*.

[2] A. Adya, W. J. Bolosky, M. Castro, G. Cermak, R. Chaiken, J. R. Douceur, J. Howell, J. R. Lorch, M. Theimer, and R. P. Wattenhofer. FARSITE: Federated, available, and reliable storage for an incompletely trusted environment. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2002)*.

[3] Y. Amir, C. Danilov, D. Dolev, J. Kirsch, J. Lane, C. Nita-Rotaru, J. Olsen, and D. Zage. Steward: Scaling byzantine fault-tolerant systems to wide area networks. In *Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN-2006)*.

[4] M. Castro, P. Druschel, A. Ganesh, A. Rowstron, and D. Wallach. Security for structured peer-to-peer overlay networks. In *Proceedings of the 5th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Dec. 2002.

[5] M. Castro and B. Liskov. Practical Byzantine Fault Tolerance. In *Proc. of the 3rd USENIX Symposium on Operating S ystems Design and Implementation (OSDI 99)*, 1999.

[6] J. Cowling, D. Myers, B. Liskov, R. Rodrigues, and L. Shrira. Hq replication: A hybrid quorum protocol for byzantine fault tolerance. In *Proceedings of the Seventh Symposium on Operating Systems Design and Implementations (OSDI)*, 2006.

[7] J. Douceur and J. Howell. Byzantine fault isolation in the farsite distributed file system. In *The 5th International Workshop on Peer-to-Peer Systems (IPTPS'06)*.

[8] A. Fiat, J. Saia, and M. Young. Making Chord robust to Byzantine attacks. In *European Symposium on Algorithms (ESA)*, 2005.

[9] M. P. Herlihy and J. M. Wing. Axioms for Concurrent Objects. In *Symposium on Principles of Programming Languages (POPL'87)*.

[10] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed cachine protocols for relieving hot spots on the world wide web. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, 1997.

[11] J. Kubiatowicz, D. Bindel, Y. Chen, S. Czerwinski, P. Eaton, D. Geels, R. Gummadi, S. Rhea, H. Weatherspoon, C. Wells, and B. Zhao. Oceanstore: an architecture for global-scale persistent storage. In *Proc. of the Ninth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2000.

[12] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2), 1998.

[13] J. Li and D. Mazières. Beyond one-third faulty replicas in byzantine fault tolerant systems. In *Proc. 4th USENIX Symposium on Networked Systems Design and Implementation (NSDI '07)*.

[14] J. R. Lorch, A. Adya, W. J. Bolosky, R. Chaiken, J. R. Douceur, and J. Howell. The smart way to migrate replicated stateful services. In *Proc. EuroSys'06*.

[15] R. Morselli. *Lookup Protocols and Techniques for Anonymity*. PhD thesis, University of Maryland, College Park, 2006.

[16] E. B. Nightingale, P. M. Chen, and J. Flinn. Speculative execution in a distributed file system. In *SOSP '05: Proceedings of the twentieth ACM symposium on Operating systems principles*.

[17] N. Preguiça, J. Lourenço, and R. Rodrigues. Byzantium: Eficient byzantine fault-tolerant database replication. Grant proposal, Sept. 2006.

[18] M. K. Reiter. The rampart toolkit for building high-integrity services. In *International Workshop on Theory and Practice in Distributed Systems*, 1995.

[19] R. Rodrigues. *Robust Services in Dynamic Systems*. PhD thesis, MIT, 2005.

[20] R. Rodrigues and B. Liskov. Rosebud: A scalable byzantine-fault-tolerant storage architecture. MIT LCS TR/932, Dec. 2003.

[21] R. D. Schlichting and F. B. Schneider. Fail-stop processors: An approach to designing fault-tolerant computing systems. *ACM Transactions on Computer Systems*, 1(3):222–238, Aug. 1983.