Replication with variable consistency and tolerance to arbitrary uncorrelated faults

Miguel Pires
miguel.pires@tecnico.ulisboa.pt

Instituto Superior Técnico (Campus Taguspark)
Av. Prof. Doutor Aníbal Cavaco Silva
2744-016 Porto Salvo - Portugal

Abstract. The abstract should summarize the contents of the paper and should contain at least 70 and at most 150 words. It should be written using the *abstract* environment.

Keywords: A few key words

1 Introduction

2 Related Work

Due to the specific characteristics of the scenario for which we try to optimize, there are several areas that contain works of interest to us. Subsection 2.1 describes previous work regarding consensus and several Paxos variants.

2.1 Paxos

The Paxos protocol family solves consensus by circumventing the well-known FLP impossibility result [2]. It does this by making the observation that, while consensus is unsolvable is asynchronous systems, a system is usually synchronous since delays are sporadic and temporary. It is for this relaxed synchrony model, Partial Synchrony, that Paxos is designed [3]. Paxos ensures consistency even when the system is asynchronous but can only guarantee progress while it is synchronous and no more than f faults occur for a system of $2f + 1$ replicas. The classic form of Paxos employs a set of proposers, acceptors and learners and runs in a sequence of ballots. To ensure progress during synchronous periods, proposals are serialized by a distinguished proposer, the leader. It is the leader that gathers proposals from other proposers and requests, in a *phase 1a* message, that acceptors promise to not accept any ballots smaller than its own. Each acceptor responds to the leader's request, in a *phase 1b* message, with either a promise or a message stating that it has already sent a promise to a proposer with a higher-numbered ballot. If the leader manages to obtain a promise from a quorum of acceptors, then it sends a *phase 2a* message containing its proposal. The acceptors may accept the proposal and inform the learners with a *phase 2b*

message, if another proposer hasn't requested a promise for a higher-numbered ballot in the meanwhile. The fact that any proposer can request a promise and prevent lower ballots from being completed is the reason why Paxos can only ensure progress in a synchronous environment. Some proposer must be able to establish itself as the only leader in order to prevent proposers from continuously requesting promises for ballots with higher numbers.

We now describe the form in which Paxos is most commonly deployed, Multi(Decree)-Paxos. Multi-Paxos provides an optimization of the basic Paxos' message pattern by omitting the first phase of messages from all but the first ballot for each leader [8]. This means that a leader only needs to send a *prepare* message once and subsequent proposals may be sent directly in *phase 2a* messages, without requesting a promise from the acceptors. This reduces the message pattern in the common case from five message delays to just three (from proposal to learning). Since there are no implications on the quorum size or guarantees provided by Paxos, the reduced latency comes at no additional cost.

Fast Paxos observes that it's possible to improve on previous results by allowing proposers to propose values directly to acceptors. To this end, the protocol distinguishes between fast and classic ballots, where fast ballots use $3f + 1$ acceptors instead of the usual $2f + 1$ [5]. The optimized scenario occurs during fast ballots, in which only two messages broadcasts are necessary: *phase 2a* messages between a proposer and the acceptors, and *phase 2b* messages between acceptors and learners. This creates the possibility of two proposers concurrently proposing values to the acceptors and generating a conflict. The cost of dealing with this conflict depends on the chosen method of recovery, coordinated or uncoordinated, and if the leader or acceptors are also learners. The leader may choose the recovery method in the beginning of the ballot. If coordinated recovery is chosen in a ballot i, then the cost is a *phase 2* of round $i+1$ (i.e., two additional message delays). On the other hand, if uncoordinated recovery is chosen then the cost is a single *phase 2b* message in round $i + 1$ (i.e., one message delay) [5]. These numbers assume that either the leader (coordinated recovery) or the acceptors (uncoordinated recovery) are learners, which is a reasonable assumption.

Generalized Paxos addresses Fast Paxos' shortcomings regarding collisions. More precisely, it allows acceptors to accept different sequences of commands as long as dependence relations are preserved [4]. Generalized Paxos abstracts the traditional consensus problem of agreeing on a single value to the problem of agreeing on an increasing set of values. *C-structs* provide this increasing sequence abstraction and allow us to define different consensus problems. Defining *c-structs* as command histories enables acceptors to agree on different sequences of commands and still preserve consistency as long as dependence relationships are not violated. This means that commutative commands can be ordered differently regarding each other but interfering commands must preserve the same order across each sequence at any learner. This guarantees that solving the consensus problem for histories is enough to implement a state-machine replicated system.

Should I go into more detail and mention compatibility and etc?

Mencius is also a variant of Paxos that tries to address the bottleneck of having a single leader through which every proposal must go through. In Mencius, the leader of each round rotates between every process. The leader of round i is the process p_k, such that $k = mod_n\ i$. Leaders with nothing to propose can skip their turn by proposing a *no-op*. If a leader is slow or faulty, the other replicas can execute *phase 1* to revoke the leader's right to propose a value but they can only propose a *no-op* in its stead [7]. Considering that non-leader replicas can only propose *no-ops*, a *no-op* command from the leader can be accepted in a single message delay since there is no chance of another value being accepted. If some non-leader server revokes the leader's right to propose and suggests a *no-op*, then the leader can still suggest a value $v \neq$ *no-op* that will eventually be accepted as long as l isn't permanently suspected. The usage of a unreliable failure detector is enough to guarantee that eventually all faulty processes, and only those, will be suspected. Mencius also takes advantage of commutativity by allowing out-of-order commits, where values x and y can be learned in different orders by different learners if there isn't a dependence relationship between them. Experimental evaluation confirms that Mencius is able of scaling to a higher throughput than Paxos.

2.2 Non-crash Fault Models

Non-crash fault models appeared to cope with the effect of malicious attack and software errors. These models assume a stronger adversary than previous crash fault models, such as arbitrary faults. Byzantine Generals Problem is defined as a set of Byzantine generals that are camped in the outskirts of an enemy city and have to coordinate an attack. Each general can either decide to attack or to retreat and there may be f traitors among the generals. The problem is solved if every loyal general agrees on what action to take [6]. Like the traitorous generals, the Byzantine adversary is one that may display arbitrary behaviour and even coordinate multiple faulty replicas.

PBFT is a protocol that solves consensus for state machine replication (SMR) while tolerating up to f Byzantine faults [1]. The system moves through configurations called *views* in which one replica is the primary and the remaining replicas are the backups. The safety property of the algorithm requires that operations be totally ordered. The protocol starts when a client sends a request for an operation to the primary, which in turn assigns a sequence number to the request and multicasts a *pre-prepare* message to the backups. This message contains information such as the timestamp, the message's digest, the view and the actual message. If a backup replica accepts the pre-prepare message, after verifying that the view number and timestamp are correct, it multicasts a *prepare* message to and adds both messages to its log. The prepare message is similar to the pre-prepare message except that it doesn't contain the client's request message. Both of these phases are needed to ensure that the requested operation is totally ordered at every correct replica. The protocol's safety property requires that the replicated service must satisfy linearizability, therefore operations must be totally ordered. After receiving a $2f$ prepare messages, a replica multicasts

Should I also mention/describe Lamport's "Leaderless Byzantine Paxos"?

a *commit* message and commits the message to its log when it has received $2f$ commit messages from other replicas. The liveness property requires that clients must eventually receive replies to their requests, provided that there are at most $\lfloor \frac{n-1}{3} \rfloor$ faults and the transmission time doesn't increase continuously. This property represents a weak liveness condition but one that is enough to circumvent Fischer, Lynch and Paterson's impossibility result [2]. A Byzantine leader may try to prevent progress by omitting pre-prepare messages when it receives operation requests from clients but backups can trigger new views after waiting for a certain period of time.

The cost of $3f+1$ replicas in a PFBT system stems from the immense power given to the adversary. The cross fault tolerance (XFT) model claims that this adversary is too strong when compared to the behavior of deployed systems. XFT assumes that machine and network faults are uncorrelated and this assumption allows it to have a cost equal to crash fault tolerant systems, $2f+1$, while still tolerating f non-crash faults. The safety condition for this model is that correct replicas commit requests in a total order, given that the system is not in anarchy. A system is in anarchy if there are any non-crash faults and the sum of crash faults, non-crash faults and partitioned replicas is larger than the fault threshold. TODO: liveness

Should I include the formal condition: $f_nc(s) > 0 \wedge f_nc(s) + f_c(s) + f_p(s) > f$?

2.3 Variable Consistency Models

2.4 Partial Synchrony

References

1. Castro, M., Liskov, B.: Practical Byzantine Fault Tolerance. Proceedings of the Symposium on Operating System Design and Implementation (February), 1–14 (1999), http://www.itu.dk/stud/speciale/bepjea/xwebtex/litt/practical-byzantine-fault-tolerant.pdf
2. Fischer, M.J., Lynch, N.A., Paterson, M.S.: Impossibility of Distributed Consensus with One Faulty Process. Journal of the ACM 32(2), 374–382 (1985), http://portal.acm.org/citation.cfm?doid=3149.214121
3. Lamport, L.: Paxos Made Simple. ACM SIGACT news distributed computing column 5 32(4), 51–58 (2001), http://portal.acm.org/citation.cfm?doid=568425.568433
4. Lamport, L.: Generalized Consensus and Paxos (April), 60 (2005), http://research.microsoft.com/apps/pubs/default.aspx?id=64631
5. Lamport, L.: Fast Paxos. Distributed Computing 19(2), 79–103 (2006)
6. Lamport, L., Shostak, R., Pease, M.: The Byzantine Generals Problem. ACM Transactions on Programming Languages and Systems 4(3), 382–401 (1982)
7. Mao, Y., Junqueira, F.P., Marzullo, K.: Mencius: Building Efficient Replicated State Machines for WANs. Proceedings of the Symposium on Operating System Design and Implementation pp. 369–384 (2008), https://www.usenix.org/legacy/events/osdi08/tech/full_papers/mao/mao.pdf
8. van Renesse, R.: Paxos Made Moderately Complex. ACM Computing Surveys 47(3), 1–36 (2011)