Vertical Paxos and Primary-Backup Replication

Leslie Lamport, Dahlia Malkhi and Lidong Zhou
Microsoft Research
lamport@microsoft.com, dalia@microsoft.com, lidongz@microsoft.com

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organization**]: COMPUTER-COMMUNICATION NETWORKS Distributed Systems

General Terms

Reliability, Algorithms, Theory

1. Introduction

Large-scale distributed storage systems built over failure-prone commodity components are increasingly popular. Failures are common in those large systems, and replication is often the solution to data reliability. A clear gap remains between the well-known consensus algorithms and the practical replication protocols in real systems: consensus algorithms such as Paxos are used mostly to maintain global configuration information, not for the actual data replication.

The gap is not accidental; the abstract models that define the classic consensus algorithms do not fully capture the requirements of real distributed systems. In the classic consensus problem, there is a single *replica group* consisting of a fixed set of n processes, at most f of which can fail. In practice, a distributed system consists of a large number of overlapping replica groups, each responsible for maintaining a subset of the system's data. Only a small number of servers (usually $f + 1$, with a small f) are deployed for each particular replica group. When replicas fail, they are replaced promptly with new ones—a procedure called *reconfiguration*—before additional failures lead to permanent data loss. The entire system has an abundance of servers, facilitating reconfiguration and new replica deployment.

We bridge the gap with two new algorithms in a family of algorithms called Vertical Paxos [1] that is derived from the Paxos consensus algorithm. Vertical Paxos allows the configuration to change in the middle of a single consensus decision, using reconfiguration directives from an *auxiliary configuration master*. The configuration master can be implemented using the replicated state machine approach with ordinary Paxos. Such a master is natural in practice for maintaining the configurations of the many replica groups in the system. The master is called upon only for reconfigurations, which should be infrequent, so a single master can serve many separate replica groups.

Our treatment has particular relevance to the *primary-backup* approach. Classic primary-backup replication protocols, common in practical distributed systems, use only $f+1$ servers to tolerate f faults. To get around the lower bound of $2f + 1$ processes needed to solve the consensus problem, a primary-backup protocol must either use an external service such as our configuration master or make more restrictive assumptions about failures. We know of no previous primary-backup protocol based on an external service that has a rigorous correctness proof. Our two Vertical Paxos algorithms lead to two provably correct primary-backup protocols. One is a rigorous formulation of existing protocols. The second is a new variation that can be useful in practice.

PODC'09, August 10–12, 2009, Calgary, Alberta, Canada.
ACM 978-1-60558-396-9/09/08.

2. From Paxos to Vertical Paxos

Like ordinary Paxos, Vertical Paxos implements a replicated state machine by executing a sequence of logically separate instances of a consensus algorithm, instance i choosing the i^{th} state machine command. The actions of different instances are easily distinguished using explicit indices, but may interleave in time due to asynchrony.

There are four roles: *clients* submit commands to be added to a totally ordered sequence of commands. *Learners* implement the state machines: they learn what command is chosen at each instance and execute the commands in order. *Leaders* and *acceptors* execute a protocol for reaching consensus decisions. Certain sets of acceptors are called *quorums*; any two quorums have a nonempty intersection. The set of acceptors and its quorum structure is called the *configuration* for the consensus. A client sends requests to any active leader. A single process can play multiple roles.

For fault tolerance, a new leader must be activated when a previous leader fails to make timely progress. As in ordinary Paxos, the execution of a consensus instance may consist of multiple leader activations, each activation attempting to reach a decision. Leader activations are called (for historical reasons) *ballots*; each ballot has a unique number. A replicated state-machine execution progresses both *horizontally* as a sequence of consensus instances and *vertically* with increasing ballot numbers.

At the heart of the consensus algorithm is a protocol performed by a leader with a new ballot. As in ordinary Paxos, a new leader must discover, for all instances, if any command was chosen at lower-numbered ballots. At the same time, the new leader must stop lower-numbered ballots from choosing any further commands. The leader then tries to ensure the choice of a command in any instance for which a command was proposed by a previous leader but possibly not chosen. For instances in which no command has been proposed, it

can propose new ones. A command is chosen at a ballot if a quorum of acceptors have acknowledged that command at this ballot. A leader of a later ballot may similarly query for the past proposed commands and block new ones by contacting a quorum of acceptors. In standard Paxos, the leader of a new ballot performs the following actions.

Phase 1: The leader performs one round of message exchanges with acceptors. When it hears back from a quorum of acceptors, it learns for every instance either (i) a command c that might have been chosen, or (ii) that no command was chosen. In this exchange, the leader also obtains a commitment from the acceptors not to respond to any further proposals in lower-numbered ballots (which in an asynchronous system could arrive at any time).

Phase 2: For every instance in which case (i) holds, the leader proposes the command c to the acceptors. For instances in which case (ii) holds, it proposes new commands as it receives them from clients.

The acceptors accept, store, and acknowledge a leader's proposal, unless they were instructed by a leader of a higher-numbered ballot to ignore this ballot.

Vertical Paxos generalizes the traditional Paxos algorithm in two ways. The first is that Vertical Paxos allows a different configuration to be associated with each ballot. A separate auxiliary configuration master decides and maintains the mapping from ballot numbers to the corresponding configurations. Reconfiguration therefore takes place "vertically", across the ballot numbers.

The second generalization is that it distinguishes *read* quorums from *write* quorums of acceptors, where any write quorum intersects with any other quorum; the first phase of the protocol involves a read quorum, while the second a write quorum. This generalization is rather simple, but becomes especially significant when the configuration changes from ballot to ballot. In particular, primary-backup replication uses configurations where read and write quorums are distinct, and reconfigures upon a failure.

In Vertical Paxos, a new ballot is started by the configuration master, which chooses its number, leader, and configuration. The two phases of Vertical Paxos work as follows. In the first phase, the leader communicates not only with a read quorum of acceptors in the configuration of the new ballot, but also with read quorums in the configurations of lower-numbered ballots. It thus learns of any commands chosen in lower-numbered ballots and, at the same time, prevents any new commands from being chosen in those ballots.

The second phase of Vertical Paxos is the same as in the standard Paxos, except that the leader contacts a write quorum in the configuration of the current ballot.

An old configuration becomes obsolete, and its read quorums no longer contacted, when all the possibly-chosen commands stored on its acceptors are known to acceptors for higher-numbered ballots. This occurs through a *state transfer* process, where a leader reads the commands stored on read quorums of lower-numbered ballots and writes the possibly-chosen commands to a write quorum in the new ballot. The two variants of the Vertical Paxos algorithm differ on when a configuration is considered *active* with respect to the state transfer.

Vertical Paxos II ensures that there is a single active configuration at any time. A new leader initiates state transfer from the current active configuration to the new one, and only after the state transfer completes does it request the master to activate the new configuration.

The Vertical Paxos II leader protocol is adapted from the Paxos leader protocol as follows. In the first phase, the leader contacts a read quorum of acceptors in the currently active ballot. The second phase is broken into several stages.

A: For every instance at which the leader learns a command c that might have been chosen, the leader proposes c to a write-quorum of acceptors in the second phase.

B: When the second phase for all those instances is completed, the leader asks the master to activate the new ballot number (and the new configuration).

C: For other instances (where no command has been chosen), the leader waits for the next proposed command from a client and proposes the value in phase 2 of the protocol.

In Vertical Paxos I, the master makes the new configuration active immediately. The configuration remains active until the master hears from a leader that the configuration's state has been transferred to the acceptors of a higher-numbered ballot. This algorithm allows multiple configurations to be active at the same time. Upon starting a new ballot, the master informs the new ballot's leader which configurations are active. Readers are referred to [1] for full descriptions of both Vertical Paxos protocols.

3. Vertical Paxos and Primary-Backup

The most interesting case of Vertical Paxos is when any single acceptor forms a read quorum and the only write quorum is the set of all acceptors. Such a configuration allows f-fault tolerance with only $f + 1$ acceptors. Moreover, we can make the leader one of the acceptors and, upon reconfiguration, choose the new leader from among the current acceptors. The new leader by itself is then a read quorum for the previous ballot, and it can perform phase 1 all by itself. This means that, in Vertical Paxos II, the only state transfer needed is between the leader and any new acceptors. With the leader as the *primary* and all other acceptors as *backups*, we obtain a traditional primary-backup system.

Most primary-backup protocols maintain a single active configuration, as captured by Vertical Paxos II, which requires state transfer before reconfiguration. In practical systems, state transfer tends to involve copying a large amount of data and is therefore costly. By allowing multiple active configurations, Vertical Paxos I decouples state transfer from reconfiguration. A new configuration can be activated to accept new requests while the state is transferred from the old configuration.

4. References

[1] Leslie Lamport, Dahlia Malkhi, and Lidong Zhou. Vertical Paxos and primary-backup replication. Technical Report MSR-TR-2009-63, Microsoft, May 2009.