Ketch: Reliability and Availability for Primary-Backup Replication

Alan Jones, watercraftsoftware@gmail.com

Abstract

Primary-backup replication has been in use for many years to provide reliability for centralized stateful applications, however, a specification of the protocol to manage such replication remains elusive. Ketch is such a protocol, designed to provide the same reliability and availability in practice as replicated log protocols such as Raft[1] and Multi-Paxos[2]. The feature of Ketch that makes this possible is the additional requirement to bound clock drift between servers, which is part of many system designs but is excluded from the log protocol models.

Features of the Ketch protocol include:

- **Strict consistency** - Values returned in any read operation on the database reflect the most recent writes.
- **Availability in the presence of server faults** - The database service is responsive as long as some majority of servers in a quorum group remain responsive.
- **Dynamic quorum groups** - Availability of smaller quorum groups is enhanced by reconfiguring the group around faults. This tolerates a larger number of server faults without the write overhead of a larger quorum as long as there is time between faults to reconfigure.
- **Durability** - When availability is compromised by unresponsive servers, service can be restored at a later time if any majority of the quorum group becomes responsive. During reconfiguration, a majority from both the old and new quorum groups is required for recovery until the new group is brought into service.

As with the replicated log protocols, Ketch tolerates message transmission failures including: propagation delay, loss, reordering, and duplication.

Ketch Components

Ketch provides reliability by completing updates only when committed to a majority of some set of servers. Let's call the group of servers among which the majority is selected a **quorum group** and the servers that comprise a majority a **data quorum**. Bounding clock drift allows us to obtain a **lease** for the primary role by agreement from a potentially different quorum from the same quorum group. Providing availability in the presence of slow or faulty servers requires reconfiguring the data quorum, identifying servers with current data and replicating that data to a new data quorum. Ketch integrates the protocol to reconfigure the data quorum with the protocol to reconfigure the quorum group. This provides many benefits of a larger quorum group size without associated overhead. When a member of the current quorum group becomes unresponsive a new quorum group composed of currently responsive servers is formed that is prepared to tolerate future failures. Let's call the servers available to participate in quorum groups **candidates** and sequences of quorum groups **epochs**. Ketch can be described as a composition of protocols:

- **Discovery** - Identify responsive servers that can participate in the next epoch.
- **Lease** - Gain the primary role for a given epoch needed to coordinate updates. The lease protocol requires a state machine called an **acceptor** which is running on each server in the quorum group.
- **Epoch** - Configure lease acceptors for a new epoch and revoke the old epoch.
- **Replica Management -** Copy data to replicas in a new data quorum and publish new epochs when the data quorum is ready.

Discovery

The goal of discovery is to identify responsive servers that can be used to form quorum groups. There are many designs that can serve this purpose and active research, but a very good analysis and design at the time of writing can be found in the SWIM protocol paper [3].

Lease

Paxos Lease [4] is a concise presentation of a distributed lease protocol. Paxos Lease is composed of two state machines: **acceptors** running in the quorum group and **proposers** that contest for a lease. Acceptor state is not required to be persisted, but doing so provides availability of the lease service for the **acceptor timeout** after acceptors restart. The acceptor state machine compares time values taken from the local clock, so persisting state requires a clock that maintains time across restarts. Paxos Lease does not require clocks from multiple servers to be synchronized, only that their drift can be bounded. The state maintained by acceptors in the lease protocol consists of:

- Accepted proposal **server ID**
- Accepted proposal **expiry time**
- Highest promised **ballot number**

Proposer state is not persisted and is associated with the replica serving data or reconfiguring the quorum group. Leases are granted in the context of an epoch where each epoch has a quorum group on which acceptors run. For Ketch we augment the lease protocol to allow the state machine that revokes an old epoch to restart and re-acquire the lease from the epoch being revoked. Lease requests are tagged with a **successor epoch ID** and, in that context, the acceptor must permit the lease protocol's prepare and propose requests for a revoked epoch only with the matching successor ID.

Epoch

Ketch manages the transitions between epochs configured on candidate servers, where the publishing of a new epoch requires revoking the previous epoch. Each epoch has the following state persisted on servers in the quorum group in addition to the state required for the lease protocol:

- Epoch ID - unique identifier for the epoch
- Successor Epoch ID - valid if revoked, empty for the current epoch
- State - **new**, **open** for new updates to the managed data or **closed**

To manage the transition between epochs, each server must maintain the above state for at most two epochs for each managed data set: the **current** one if set up and one previous **revoked** epoch if present. On a request to set up an epoch the current one is replaced. On a request to revoke an epoch, the current one is reassigned as revoked if the ID matches and/or the revoked epoch is created or replaced with the values in the request.

Replica Management

The following state is persisted by Ketch with each copy of the managed data:

- Instance ID - unique identifier for the data being managed
- Epoch ID - unique identifier for the epoch
- Epoch Quorum Group - list of server IDs for the epoch's quorum group
- Epoch Data Quorum - list of server IDs for the current data quorum
- Prior Epoch ID - valid if prior epoch is not yet revoked
- Prior Epoch Quorum Group - list of server IDs for the prior epoch's quorum group
- State - state of this replica's data with respect to the data quorum, either **in-sync** or **catch-up**
- Peer State - summary state for other members of the data quorum, either in-sync or catch-up

The job of the replica management protocol for an in-sync replica is to monitor the health of servers in it's quorum group and reconfigure the group when these servers become unresponsive.

Reconfiguring the Quorum Group

Reconfiguring the quorum group consists of the following steps with a discussion of each to follow:

1. Choose a new epoch ID with a quorum group and data quorum from healthy candidates
2. Persist new epoch metadata together with metadata for the prior epoch
3. Obtain lease for the prior epoch
4. Mark prior epoch as closed for updates
5. Replicate data to the new data quorum
6. Obtain lease for the new epoch
7. Mark replicas in the new data quorum as in-sync
8. Revoke the prior epoch
9. Mark the new epoch as open for update and begin serving data

Step 1: Choose a New Epoch

The new epoch membership is chosen to maximize overlap between members of the existing epoch's data quorum and the list of healthy nodes reported by the discovery protocol. The epoch ID must be globally unique, which can be achieved with a vector of the local server ID and a sequence number or through random selection over a large namespace.

Step 2: Persist New Epoch

The replica metadata listed at the start of this section must be persisted for the new epoch in the same fault domain as each replica's data. In this step we update the prior epoch from the previous current epoch only if prior epoch is not set. An empty prior epoch field indicates that the prior epoch was successfully revoked. Once the prior epoch is revoked (Step 8) and if we are unable to open the new epoch (Step 9), the new epoch can be abandoned and another one constructed.

Step 3: Obtain Lease for Prior Epoch

Using the same instance ID and lease protocol that we will use for the primary role during replication, we obtain the prior epoch's lease through message exchanges with the prior epoch's quorum group. Again, the quorum used to obtain the lease is potentially different from the data quorum used in replication. For example, with a quorum group of three servers we only need to replicate to a data quorum of two for availability while the third acts as "witness" by virtue of participating in the lease protocol. We augment the lease protocol by providing the successor's epoch ID when requesting a lease and by returning an "open" or "closed" state for the epoch from the quorum of acceptors that grant the lease. The epoch is considered closed if any of the acceptors return a state other than “open”.

Step 4: Close the Prior Epoch

Closing the epoch signals to any server that acquires the lease that this epoch can only be used to replicate data into a new epoch and not for serving updates. The protocol to close the epoch involves a request and response to a quorum of acceptors within the prior epoch. The requesting server keeps track of which server IDs have responded and can proceed to the next step only after a quorum is achieved. When acquiring the lease, if any acceptor reports the epoch as closed it must be treated as such.

Step 5: Replicate Data to New Data Quorum

This step involves copying data to a new data quorum and configuring synchronous replication such that new updates will block until all members of the data quorum have them. Note that the new epoch should be configured in the closed state until the final step.

Step 6: Obtain Lease for New Epoch

Use the Paxos Lease protocol to obtain a lease for the new epoch. The lease prevents new replicas from reconfiguring the quorum group before this replica has had a chance to complete the current reconfiguration.

Step 7: Mark New Replicas as In-Sync

When constructing a new replica it is marked as catch-up and the local replica is marked in-sync with peer state catch-up until replication is complete. This step involves request and response messages to all replicas in the new data quorum to set their state and peer state to in-sync.

Step 8: Revoke Prior Epoch

We revoke the prior epoch with request and response messages to any quorum within the prior epoch’s quorum group. The request carries the successor epoch’s ID, which is used by the acceptor to grant the lease again (Step 3) should any replica in the successor epoch with in-sync data attempt to restart this process. A matching successor ID is not required for the request to revoke itself. Multiple successor epochs can be formed and attempt to revoke the prior epoch, but when a quorum of prior epoch acceptors grant the revoke request only one successor epoch can reacquire the lease to complete a reconfiguration.

Step 9: Mark New Epoch as Open

When a quorum of prior epoch acceptors has acknowledged that it is revoked, the new epoch can be enabled for updates. This is done with request and response messages with the entire quorum group of the new epoch. As the new epoch was chosen on advice from the discovery protocol it is expected that all of these servers are still responsive. If not, a new reconfiguration is started. During the new reconfiguration, the epoch that we were unable to open remains closed and becomes the prior epoch. When all servers in the new epoch have responded to a request to open, this replica can begin serving updates in the primary role under the term of the new epoch lease.

Replica Management State Machine

The sequence of steps to reconfigure the quorum group can be restarted at several points by assessing the current state. This can be viewed as a sequence of questions asked periodically by servers in the candidate pool with appropriate responses:

1. Do I have replica state? If not, exit.
2. Is my replica state in-sync? If not, exit.
3. Are any servers in my quorum group unresponsive? If so, engage from Step 1.
4. Do I have a prior epoch ID and quorum group with no lease?
   If so, obtain the prior epoch's lease and close the prior epoch (Steps 3 and 4).
5. Are my peer members of the data quorum in-sync? If not, engage from Step 5.
6. Do I already have a lease for this epoch? If not, engage from Step 6.
7. Do I have a prior epoch ID and quorum group? If so, engage from Step 8.
8. Is this epoch open? If not, engage from Step 9.
9. If unable to open the new epoch, engage from Step 1 with this as the prior epoch ID.

Reliability

Reliability in the context of primary-backup replication can be viewed as a combination of the consistency and durability properties and rephrased as:

- If a quorum of servers remains responsive for long enough, the protocol will permit serving data and all reads will reflect the content of the most recent writes.

In Ketch, when the quorum group is not being reconfigured the epoch's data quorum must be responsive to serve data. Should one or more servers in the data quorum fail, a new quorum group must be established. For that some other quorum of servers from the previous epoch must be responsive and it is the overlap between this quorum and the data quorum that will drive the reconfiguration, i.e. the responsive replicas that have current data. During reconfiguration there may be more than one such replicas vying for successor to the prior epoch, however, only one at a time will be able to obtain the prior epoch's lease. If the replica that currently has the prior epoch lease is able to perform a revoke on a majority of acceptors in the prior epoch (Step 8), then it's new epoch becomes the only successor that can obtain the prior epoch's lease. At Step 8 we have replicated data to the data quorum in the new epoch (Step 5), so any of these servers can reacquire the old epoch's lease using the new epoch's ID (Step 3) to continue the process. This requires a quorum of servers from both the new and old epoch's, but not necessarily data quorum servers, fulfilling the general statement of reliability above.

A typical configuration of the replicated log protocols has a quorum group of 3. This configuration provides both availability should one server become unresponsive and reliability should one server be lost altogether, for example, by failure of the storage media. Ketch cannot match this combination of reliability and availability because to serve new updates after a member of the quorum becomes unresponsive a new epoch must be configured. With only 2 responsive servers remaining, the new epoch cannot provide fault tolerance. To provide these properties with Ketch, configure a larger pool of candidates. For example, with a quorum group of size 3 and a candidate pool of 4 Ketch can reconfigure the quorum group without compromising reliability or availability. Similarly, a Ketch candidate pool of 7 can match the properties of a replicated log quorum group of 5, with the additional servers acting as spares to complete the original quorum group size during failures.

Availability

Availability of any distributed system design is tightly coupled to the real-time behavior of the systems. A key component in Ketch involves the ability of the current primary to detect that its lease has expired and stop communicating to clients. This time must be bounded for another server to safely acquire the lease and serve data. Similarly, the fault detection time of the Discovery protocol has a direct impact on the **failover time** needed to begin serving updates again after the primary halts.

Ketch introduces a potentially large component into the failover time compared with the replicated log protocols when data must be copied to new replicas. This can be mitigated by the use of **asynchronous replicas**. An asynchronous replica is one on which the primary server sends new updates as they arrive but does not wait for the replica to respond before returning the status to clients. These replicas can be treated as permanently in the catch-up state and can never be used to create new replicas. However, if the application supports it, they can be quickly converted to in-sync replicas during epoch reconfiguration. For example, in our 4 candidate deployment with a quorum group size of 3, the data quorum would be 2 and the third quorum group member could be configured as an asynchronous replica. When one of the members of the data quorum fails, the asynchronous replica can be promoted to construct a new 2 server data quorum by copying the minimal set of updates to bring it up-to-date.

Conclusion

With the additional assumption of bounded clock drift and some spare servers to complete a new quorum, Ketch can provide the reliability and availability of replicated log protocols without sacrificing the read performance of a centralized server.

References

1. D. Ongaro and J. Ousterhout, Stanford University, In Search of an Understandable Consensus Algorithm. https://raft.github.io/raft.pdf, 2014.
2. L. Lamport. Paxos made simple. http://research.microsoft.com/en-us/um/people/lamport/pubs/paxos-simple.pdf, 2001.
3. Das, A., Gupta, I., Motivala, A.: SWIM: Scalable Weakly-consistent Infection-style Process Group Membership Protocol. https://www.cs.cornell.edu/~asdas/research/dsn02-swim.pdf. 2002.
4. M. Trencseni, A. Gazso, and H. Reinhardt. PaxosLease: Disk-less Paxos for Leases. https://arxiv.org/pdf/1209.4187.pdf, 2012.

Message Catalog

In addition to the content listed below, all messages should carry an instance_id to identify the data being managed.

Catagory	**Type**	**Sub-Type**	**Source**	**Destination**	**Content**
Lease	Prepare	Request	Replica	Acceptor	source_server_id, epoch_id, ballot
Lease	Prepare	Response	Acceptor	Replica	source_server_id, epoch_id, ballot, highest_promised_ballot
Lease	Propose	Request	Replica	Acceptor	source_server_id, epoch_id, ballot, server_id, timeout, successor_epoch_id
Lease	Propose	Response	Acceptor	Replica	source_server_id, epoch_id, ballot, epoch_state
Epoch	Setup	Request	Replica	Acceptor	source_server_id, epoch_id
Epoch	Setup	Response	Acceptor	Replica	source_server_id, epoch_id
Epoch	Open	Request	Replica	Acceptor	source_server_id, epoch_id
Epoch	Open	Response	Acceptor	Replica	source_server_id, epoch_id
Epoch	Close	Request	Replica	Acceptor	source_server_id, epoch_id
Epoch	Close	Response	Acceptor	Replica	source_server_id, epoch_id
Epoch	Revoke	Request	Replica	Acceptor	source_server_id, epoch_id, successor_epoch_id
Epoch	Revoke	Response	Acceptor	Replica	source_server_id, epoch_id, successor_epoch_id
Replica	Create	Request	Replica	Replica	source_server_id, epoch_id, quorum_group, data_quorum, prior_epoch_id, prior_epoch_quorum_group
Replica	Create	Response	Replica	Replica	source_server_id, epoch_id
Replica	UpdateInSync	Request	Replica	Replica	source_server_id, epoch_id
Replica	UpdateInSync	Response	Replica	Replica	source_server_id, epoch_id