**Leader Election**

Leader – Assigning unique powers to one, for example to write to a shared resource or blob store. Elect a leader among a set of candidate processes or nodes.

Distributed locks can help elect a leader. Zookeeper implements it practically.

Why does one need distributed locks?

* Efficiency: Prevent doing the same work twice. If the lock fails and two nodes end up doing the same piece of work, the result is a minor increase in cost
* Correctness: If the lock fails and two nodes concurrently work on the same piece of data, the result is a corrupted file, data loss, permanent inconsistency

So technically, distributed locks help elect a leader for that task. The one which acquires the lock becomes the leader and can write to the file. That’s the philosophy.

Just acquiring lock with support for lease time is not enough. Use case for fencing token:

Diagram

Description automatically generated

In Zookeper, znode’s version number can act as the fencing token.

How locking works in Zookeper

Clients wishing to obtain a lock do the following:

1. Call **create( )** with a pathname of "\_locknode\_/lock-" and the *sequence* and *ephemeral* flags set.
2. Call **getChildren( )** on the lock node *without* setting the watch flag (this is important to avoid the herd effect).
3. If the pathname created in step **1** has the lowest sequence number suffix, the client has the lock and the client exits the protocol.
4. The client calls **exists( )** with the watch flag set on the path in the lock directory with the next lowest sequence number.
5. if **exists( )** returns false, go to step **2**. Otherwise, wait for a notification for the pathname from the previous step before going to step **2**.

The unlock protocol is very simple: clients wishing to release a lock simply delete the node they created in step 1.

Other use cases of Zookeper: Service Discovery – The instances register themselves in the zookeeper with ephemeral nodes. Zookeper can detect the crash of the service instances and notify the watcher. Kafka – Leader of a Partition election.

How leader election works in Zookeper:

A simple way of doing leader election with ZooKeeper is to use the **SEQUENCE|EPHEMERAL** flags when creating znodes that represent "proposals" of clients. The idea is to have a znode, say "/election", such that each znode creates a child znode "/election/n\_" with both flags SEQUENCE|EPHEMERAL. With the sequence flag, ZooKeeper automatically appends a sequence number that is greater that any one previously appended to a child of "/election". The process that created the znode with the smallest appended sequence number is the leader.

That's not all, though. It is important to watch for failures of the leader, so that a new client arises as the new leader in the case the current leader fails. A trivial solution is to have all application processes watching upon the current smallest znode, and checking if they are the new leader when the smallest znode goes away (note that the smallest znode will go away if the leader fails because the node is ephemeral). But this causes a herd effect: upon of failure of the current leader, all other processes receive a notification, and execute getChildren on "/election" to obtain the current list of children of "/election". If the number of clients is large, it causes a spike on the number of operations that ZooKeeper servers have to process. To avoid the herd effect, it is sufficient to watch for the next znode down on the sequence of znodes. If a client receives a notification that the znode it is watching is gone, then it becomes the new leader in the case that there is no smaller znode. Note that this avoids the herd effect by not having all clients watching the same znode.

More details: <https://bikas-katwal.medium.com/zookeeper-introduction-designing-a-distributed-system-using-zookeeper-and-java-7f1b108e236e>